UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUACAO EM CIÊNCIA DA COMPUTAÇÃO

PROPOSTA DE UM VOCABULÁRIO SEMÂNTICO PARA DESCOBRIR SERVIÇOS NA INTERNET DAS COISAS

MAYKA DE SOUZA LIMA

SÃO CRISTÓVÃO/SE
2016

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL
UNIVERSIDADE FEDERAL DE SERGIPE

Lima, Mayka de Souza
L732p Proposta de um vocabulário semântico para descobrir serviços na internet das coisas / Mayka de Souza Lima; orientador Admilson de Ribamar Lima Ribeiro. - São Cristóvão, 2016.
123 f.: il.

Dissertação (Mestrado em Ciência da Computação) - Universidade Federal de Sergipe, 2016.

1. Semântica - Análise de redes. 2. Web semântica. 3. Serviços da web. 4. World wide web (Sistema de recuperação da informação). 5. Ferramentas de busca na web. 6. Computação semântica. l. Ribeiro, Admilson de Ribamar Lima, orient. ll. Título.

CDU 004.774.2

MAYKA DE SOUZA LIMA

PROPOSTA DE UM VOCABULÁRIO SEMÂNTICO PARA DESCOBRIR SERVIÇOS NA INTERNET DAS COISAS

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação (PROCC) da Universidade Federal de Sergipe (UFS) como parte dos requisitos para obtenção do título de Mestre em Redes de Computadores e Sistemas Distribuídos.

SÃO CRISTÓVÃO/SE

MAYKA DE SOUZA LIMA

PROPOSTA DE UM VOCABULÁRIO SEMÂNTICO PARA DESCOBRIR SERVIÇOS NA INTERNET DAS COISAS

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação (PROCC) da Universidade Federal de Sergipe (UFS) como parte dos requisitos para obtenção do título de Mestre em Redes de Computadores e Sistemas Distribuídos.

BANCA EXAMINADORA

Prof. Dr. Admilson de Ribamar Lima Ribeiro, Orientador
Universidade Federal de Sergipe (UFS)

Prof. Dr. Edward David Moreno, Coorientador
Universidade Federal de Sergipe (UFS)

Profa Dra. Edilayne Meneses Salgueiro
Universidade Federal de Sergipe (UFS)

Prof. Dr. Sergio Takeo Kofuji, Membro
Escola Politécnica da Universidade São Paulo (POLI-USP)

PROPOSTA DE UM VOCABULÁRIO SEMÂNTICO PARA DESCOBRIR SERVIÇOS NA INTERNET DAS COISAS

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação (PROCC) da Universidade Federal de Sergipe (UFS) como parte dos requisitos para obtenção do título de Mestre em Redes de Computadores e Sistemas Distribuídos.

Trabalho aprovado, São Cristóvão - SE, 31 de agosto de 2016.

_____

Prof. Dr. Admilson de Ribamar Lima Ribeiro,
Orientador - Universidade Federal de Sergipe (UFS)

_____

Prof. Dr. Edward David Moreno
Coorientador - Universidade Federal de Sergipe (UFS)

_____

Profa. Dra. Edilayne Meneses Salgueiro
Universidade Federal de Sergipe (UFS)

_____

Prof. Dr. Sergio Takeo Kofuji
Escola Politécnica da Universidade São Paulo (POLI-USP)

AGRADECIMENTO

Ao meu bom Deus, guias, mentores e anjos protetores, pela vida e a possibilidade de empreender esse caminho evolutivo, por propiciar tantas oportunidades de estudos e por colocar em meu caminho pessoas amigas e preciosas.

A MINHA FAMÍLIA, especialmente aos meus irmãos Maisa e Gledson, a meus cunhados Adriana e Pedro, pela força nos momentos mais difíceis.

AOS AMIGOS do Mestrado que compartilharam comigo esses momentos de aprendizado e nos ajudamos mutuamente.

AOS AMIGOS, Othon, Luciana, Ana Paula, Vanessa, Valdivia, Joelma, Kamilla, Maria Virgínia e AOS MEUS ALUNOS DO IFS CAMPUS ITABAIANA, que mesmo seguindo caminhos diversos, sempre se fizeram presentes com lembranças, palavras de encorajamento e amor.

AO MEU ORIENTADOR Prof. Dr. Admilson de Ribamar Lima Ribeiro e CO-ORIENTADOR Prof. Dr. Edward David Moreno Ordonez, um agradecimento carinhoso por todos os momentos de paciência, compreensão e competência.

AO PROGRAMA DE PÓS-GRADUAÇÃO DE CIÊNCIA DA COMPUTAÇÃO DA UFS, representado pelo Prof. Dr. Rogério Patrício Chagas do Nascimento, pelos momentos partilhados, sem esmorecimento e a todos os professores que fizeram parte desses caminhar.

Enfim, a todos aqueles que de uma maneira ou de outra contribuíram para que este percurso pudesse ser concluído.

RESUMO

A *Internet* das coisas é uma revolução tecnológica que conecta aparelhos eletrônicos como eletrodomésticos ou meios de transporte à *Internet*, baseando-se em protocolos de comunicação, domínios e aplicações conectando objetos físicos, como sensores ou dispositivos remotos através de comunicações sem fio. Para que objetos inteligentes realizem suas funções quando conectados à *Internet* utilizando os recursos da *web*, é necessário que as interfaces de comunicações possam tratar a busca de um determinado serviço solicitado pelo usuário com precisão. A busca deste serviço dá-se com uma semântica clara e objetiva utilizada no vocabulário para o processo de solicitação. Em algumas pesquisas citadas no trabalho a utilização dos vocabulários semânticos em serviços da *web*, apresenta alguns problemas que persistem na detecção clara destes serviços. Estes trabalhos que realizam busca de serviços foram analisados e correlacionados seus pontos positivos e negativos sobre a semântica utilizada no vocabulário e sua aplicação na IoT. Desta forma, para obter um processo de descobrimento de serviços, este trabalho apresenta um vocabulário semântico que realiza a descoberta de serviços na *Internet* das coisas. O vocabulário construído foi implementado em uma aplicação que realiza a comunicação de uma aplicação de Melhor Preço, onde tem um servidor de mercado e uma geladeira buscando um serviço na *Internet*. Após a implementação foi simulado o processo de busca do vocabulário no *Cooja Simulator* do sistema operacional *Contiki* e utilizado a linguagem JSON (*JavaScript Object Notation*) como base do código de descobrimento, uma linguagem simples e muito utilizada para aplicações *web*. Realizada a simulação analisou-se a comunicação dos serviços trocados entre o servidor Melhor Preço e as aplicações Mercado e Geladeira, obtendo que as memórias dos nós sensores não conseguiram realizar o envio da resposta do vocabulário semântico com os serviços devido uma limitação do tamanho da memória ROM (*Read Only Memory*) do nó sensor simulado.

Palavras-chave: Vocabulário. Descoberta. Semântica. Simulação. Serviços.

ABSTRACT

The Internet of Things is a network infrastructure that is based on communication protocols, domains and applications connecting physical objects, such as sensors or remote devices through wireless communications. In order for smart objects perform their functions when connected to the Internet using web resources, it is necessary that the communication interfaces can treat the search for a specific service requested by the user accurately. The search for this service gives with a clear and objective semantics used in the vocabulary for the request process. In some research cited in the work the use of semantic vocabularies in web services, has some problems that persist in the clear detection of these services. These papers that perform search services were analyzed and correlated their positive and negative aspects about the semantics used in the vocabulary and its application in the IoT. In this manner, to achieve a process of discovery services, this paper presents a semantic vocabulary that performs discovery services on the Internet of things. The vocabulary built was implemented in an application that performs the communication of an application of best price, which has a market server and a fridge that seek a particular service. After implementation the vocabulary search process was simulated in Cooja Simulator Contiki operating system and used the language JSON (JavaScript Object Notation) based on the discovery code, simple language and widely used for web applications. After the simulation, we analyzed the communication services exchanged between the Best Price server and Market applications and fridge, getting the memories of sensor nodes failed to perform the sending of the semantic vocabulary response services because of a size limitation of ROM (Read Only memory) of the simulated sensor nodes.

Keywords: Vocabulary. Discovery. Semantics. Simulation. Services.

LISTAS DE ILUSTRAÇÕES

LISTAS DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface* - Interface de Programação de Aplicações
DAML	*DARPA Agente Mark-up Language* – Agente de Linguagem de Marcação
DARPA	*Defense Advanced Research Projects Agency* - Agência de Projetos de Pesquisa Avançada de Defesa
FOAF	*Friend of a Friend*– Amigo de um amigo
JSON	*JavaScript Notation Location* – Notação de Objetos JavaScript
jUDDI	*Java Universal Description, Discovery and Integration* – Descrição Universal Java de Descoberta e Integração
OIL	*Ontology Interchange Language* – Linguagem de Intercâmbio de Ontologia
OWL	*Web Ontology Language* – Linguagem de Ontologia *Web*
OWL-S	*Semantic Markup for Web Services* – Serviços *Web* para Marcação Semântica
NIC	*National Intelligence Council* - Conselho Nacional de Inteligência
REST	*Representational State Transfer* - Transferência de Estado Representacional
RFID	*Radio Frequency Identification* - Identificação de Rádio Frequência
RSSF	Redes de Sensores Sem Fio
SAWSDL	*Semantic Annotations for WSDL and XML Schema* - Anotações semânticas para WSDL e XML Schema
SLP	*Service Location Protocol* – Protocolo de Localização de Serviço
SOA	*Service-Oriented Architecture* - Arquitetura Orientada a Serviço
SWRL	*Semantic Web Rule Language* – Linguagem de Regras da *Web* Semântica
XHTML	*eXtensible Hypertext Markup Language* – Linguagem Extensível de Marcação de Hipertexto
XML	*Extensible Markup Language* - Linguagem de Marcação de Hipertexto
WADL	*Web Application Description Language* – Linguagem de Descrição Aplicação *Web*
WSAN	*Wireless Sensor and Actuator Network* – Redes de Sensores Sem Fio e Atuador
WSMO	*Web Service Modeling Ontology* – Ontologia de Modelagem de Serviço *Web*
WSN	*Wireless Sensor Network* – Redes de Sensores Sem Fio

SUMÁRIO

1 INTRODUÇÃO

A *Internet* das coisas concentra-se na variedade de protocolos, domínios e aplicações que abrangem um conceito, revelando alta conectividade entre objetos, criando uma rede inteligente e seu conjunto é composto de entidades inteligentes. A ideia de conectar os aparelhos à *Internet* facilita não apenas projetos de edifícios inteligentes, por exemplo, como também torna possível o monitoramento remoto de outros dispositivos, desde medidores de água e energia, sensores de meio ambiente, até implantes médicos. Segundo Atzori, Iera e Morabito (2010, p.2788), a *Internet* das coisas (ou *Internet of Things*) surge como um paradigma que até 2025 dominará o mundo computacional com a interação de objetos físicos embarcados a sensores e atuadores, conectados por redes sem fio e comunicando-se usando a *Internet*. Com isso, molda-se uma rede de objetos inteligentes capaz de realizar diversos processamentos dos dados, capturar variantes do meio ambiente e reagir a estímulos por sensores externos. Esses objetos interconectados entre si com outros recursos (físicos ou virtuais), podem ser controlados através da *Internet* permitindo o surgimento de uma série de aplicações que beneficiarão novos tipos de dados, serviços e operações disponíveis.

A IoT (*Internet of Things*) visa realizar a unificação de tudo em nosso mundo sob uma infraestrutura comum, dando-nos não só o controle das "coisas" ao nosso redor, mas também mantendo-nos informados sobre o seu estado. É uma das principais tecnologias emergentes que contribuem para concretizar novos domínios de aplicação das tecnologias de informação e comunicação (TICs), a exemplo do domínio de cidades inteligentes ou domótica, no qual o uso de tecnologias avançadas de comunicação e sensoriamento visa prover serviços de valor agregado para os órgãos administrativos de tais cidades e para seus cidadãos (ZANELLA, et al.,, 2014, p. 23).

De forma geral, a IoT adquirirá espaço principalmente no moderno cenário das comunicações sem fio, difundindo uma variedade de coisas ou objetos ao nosso redor, como, por exemplo, etiquetas RFID - *Radio Frequency IDentification* (identificação por radiofrequência), telefones celulares inteligentes ou redes de sensores sem fio (RSSF). A rastreabilidade e acessibilidade usando as RFID's para infraestrutura e arquitetura da IoT, protocolos de comunicação para dispositivos inteligentes, sensores de redes (móveis),

middleware, segurança e privacidade, e muitos outros tem sido escopo de pesquisas durante os últimos anos. Entre estes se encontram o desenvolvimento computacional da semântica que manifesta o seu potencial para lidar com os problemas de heterogeneidade e interoperabilidade expostos pelo grande número de coisas conectadas com características diferentes (WANG, et al., 2012, p.1795).

Algumas aplicações que publicam os dados dos sensores e os ligam aos recursos da *web* descobertos (BARNAGHI, et al., 2010, p.2), e o processo de descoberta de serviços de sensores através de uma interface padrão (PSCHORR, et al., 2010, p.3) utilizam as tecnologias da *web* semântica. Segundo Bauer, Meissner, et al (2011, p.950), uma modelagem semântica é abordada para diferentes componentes na IoT, mecanismos automatizados de associação com entidades físicas e os dados podem ser descobertos através da pesquisa semântica e mecanismos de raciocínio, onde prevê uma infinidade de objetos heterogêneos e interações com o meio ambiente físico.

Segundo Serrano (2015, p.4), o projeto OpenIoT desenvolve e fornece uma plataforma de código aberto para a IoT, onde possibilita a interoperabilidade semântica de serviços da *internet* das coisas na nuvem. O projeto se encontra na W3C *Semantic de Sensor Networks* (SSN), que fornece um padrão comum baseado em um modelo para a representação de sensores físicos e virtuais. Além disso, oferece uma ampla gama de ferramentas visuais que permitam o desenvolvimento e implantação das aplicações da IoT com quase zero de programação. Este projeto reforça vocabulários existentes para sensores e objetos conectados à *internet*, com conceitos adicionais relevantes para IOT com integração em nuvem.

Isto coincide com o princípio do SOA (*Service-Oriented Architecture*), arquitetura orientada a serviço, um paradigma para organização e utilização de recursos distribuídos que podem estar sob o controle de diferentes domínios, onde as entidades (pessoas e organizações) criam capacidades para resolver ou apoiar uma solução para os problemas que enfrentam no decorrer da sua atividade (MACKENZIE, et al., 2006, p.5).

Geralmente as descrições possuem aspectos como as funções e os requisitos técnicos, restrições e políticas relacionadas, além de mecanismos de acesso ou resposta. Estas descrições precisam estar em uma forma (ou pode ser transformado em um formulário) no

qual sua sintaxe e semântica são amplamente acessíveis e compreensíveis. Estes conceitos de visibilidade, a interação, e efeito aplicam-se diretamente aos serviços do mesmo modo como estes foram descritos para o SOA. A visibilidade é promovida através da descrição de serviço que contém a informação necessária para interagir com o serviço e descreve isso em termos tais como as entradas de serviço, saídas e semânticas associadas. A descrição do serviço também transmite o que é realizado quando o serviço é invocado e as condições de utilização do serviço. Em geral, estas entidades oferecem capacidades e atuam como prestadores de serviços, e aqueles que fazem uso de serviços são referenciados como consumidores de serviços. A descrição desse serviço permite que os potenciais consumidores possam decidir se o serviço é adequado para as suas necessidades atuais e estabelece quando um consumidor satisfaz todos os requisitos do prestador de serviços (MACKENZIE, et al., 2006, p.6).

Os métodos existentes para descoberta de serviços e composição pode facilmente acessar os serviços bases da IoT para criar um contexto de serviços e aplicações com bases personalizadas. A modelagem semântica para o domínio da IoT fornece uma base para interoperabilidade entre os diferentes sistemas e aplicações. Segundo Wang, et al (2012, p.1795), apresentam uma descrição de ontologias para o domínio da IoT, integrando e estendendo o trabalho existente em conceitos de modelagem sobre a *Internet* das coisas. A ontologia ajuda a explorar a sinergia dos esforços existentes e fornece suporte para tarefas cruciais como recursos utilizados para a IoT, a descoberta e testes dos serviços.

1.1 Problemática e Hipótese

Para que esta conectividade entre a *internet* das coisas e os dispositivos seja realizada é necessário que as informações das aplicações sejam extraídas a partir de vocabulários semânticos específicos para sua comunicação. Este ponto de vista sugere os serviços sendo acessíveis ao público para tornar a *web* uma biblioteca de *software* global, com serviços e componentes prontos para o seu uso.

Padrões como OWL-S (*Web Ontology Language For Services*) ou WSMO (*Web Service Modeling Ontology*) permitem a construção de descrições de serviços semânticos

para executar operações de execução, de descoberta e composição de uma forma automatizada (VILLAMOR, 2010, p.5).

A fim de melhorar a integração da arquitetura dos serviços com a arquitetura REST (*Representational State Transfer*) para a *web*, iniciativas como WADL (*Web Application Description Language*) tentam habilitar serviços *web* semânticos com abordagens no estilo arquitetural RESTful (HANDLEY, 2006, p.25). Ambos, permitem aos agentes executar tarefas automáticas como a descoberta de serviços, mas descrever os serviços na *web* semântica é uma tarefa muito demorada, necessitando que algumas iniciativas como o WSMO ou microserviços fossem adotados.

Um dos principais problemas com a IoT é que seu conceito é vasto e um dos esforços é a capacidade de transmitir dados para a nuvem em um maneira de desempenho escalável e alto, enquanto, ao mesmo tempo que proporciona os meios para gerenciamento de aplicativos e fluxos de dados. Mas, essa arquitetura não fornece semântica de interoperabilidade em todas as aplicações da IoT que têm sido desenvolvida ou implantada de forma independente. Portanto, ainda não existe uma maneira fácil de combinar dados e serviços a partir de diversas aplicações da IoT que caracterizam semânticas incompatíveis (SERRANO, 2015, p.7).

Para que a IoT funcione deve possuir uma variedade de sensores, rede comunicações e tecnologias de computação. Mas quando se começa a montar diferentes tipos de tecnologias, o problema da interoperabilidade surge. Segundo Gigli e Koo (2011, p.27), propõe adotar os padrões do SOA ou a integração dos serviços *web* nas redes de sensores sem fio com o uso de *gateways* otimizados na IoT.

É necessário compreender os tipos de serviços disponíveis, para realizar a definição de um vocabulário semântico que seja capaz de identificá-los em um sensor de rede conectado na IoT. As pesquisas citadas a respeito visam à interoperabilidade entre dispositivos heterogêneos servindo diversos domínios de aplicação sensíveis ao contexto, como descoberta e gerenciamento de dispositivos.

A hipótese levada em conta nesta dissertação de mestrado é a construção de um vocabulário que possa identificar serviços na estrutura de uma rede, capaz de agilizar a

resposta ao usuário minimizando o consumo de energia e memória ao ter uma aplicação *web* conectada a qualquer dispositivo inteligente na IoT.

1.2 Objetivos

O objetivo principal da dissertação é construir um vocabulário semântico com característica de autodescoberta de serviços para a IoT realizando uma análise de desempenho proposta no estudo de caso.

Para alcançar o objetivo do trabalho, são necessários alguns objetivos específicos:

- ✓ Buscar os vocabulários utilizados em aplicações para descobertas de serviços na *web* semântica relacionados com a IoT;
- ✓ Levantar dentro das principais ferramentas utilizadas para criação de vocabulários semânticos qual melhor se aplica contexto atual nas aplicações multiplataforma que utilizam a IoT;
- ✓ Levantar quais tipos de serviços será utilizado na simulação de desempenho do vocabulário semântico definido;
- ✓ Construir o vocabulário semântico proposto para descobrir serviços na IoT;
- ✓ Analisar o desempenho do vocabulário construído em um *software* de simulação;
- ✓ Avaliar qualitativa e quantitativamente o vocabulário definido para a descoberta dos serviços na IoT.

1.3 Justificativa

Ao realizar um estudo sobre os vocabulários semânticos para a IoT, observou-se que algumas ferramentas como a OWL-S, e linguagens como WSMO e a WSML, são utilizadas para os serviços da *web* semântica. Tais ferramentas e linguagens não retratam uma análise de desempenho na busca de um recurso ou serviço na *web*. A necessidade de construir um vocabulário semântico com característica de autodescoberta surgiu para minimizar o tempo de busca da solicitação entre a aplicação e um servidor de serviços. Logo, será possível

analisar os resultados alcançados, avaliar as tecnologias utilizadas e os recursos para a construção deste vocabulário aos diversos sensores inteligentes.

A construção do vocabulário proposto irá possibilitar a realização de uma análise de desempenho que demonstrará a eficácia das classes e objetos como um todo ou parte dele para o descobrimento de um serviço na IoT. Logo, apoiam eventualmente a tomada de decisão para estabelecer processos na definição de sensores inteligentes que possam criar um ambiente absolutamente simples, aplicável e eficaz no desenvolvimento de tais tecnologias para as diversas aplicações na IoT e *web* semântica.

1.4 Organização

O trabalho segue com mais 6 (seis) capítulos: o Capitulo 2 apresenta a fundamentação teórica dos assuntos trabalhados nessa dissertação, são eles: principais conceitos da IoT e seus serviços; vocabulário semântico; ontologias; anotações semânticas; arquitetura REST; computação pervasiva e autonômica; o Capitulo 3 descreve os trabalhos correlatos explicando os principais requisitos de um vocabulário semântico utilizados para aplicações e serviços na IoT; o Capitulo 4 definirá a proposta de um vocabulário semântico para descobrir serviços na IoT; o Capítulo 5 apresentará a implementação do vocabulário semântico definido para o descobrimento dos serviços na IoT; e o Capítulo 6 serão apresentadas as conclusões e indicações de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, serão apresentados os conceitos para embasamento teórico a serem utilizados no decorrer dessa dissertação. Em um primeiro momento, são descritos conceitos sobre a IoT, sua utilização no mundo virtualmente aplicado hoje na *web*, os serviços e as descobertas desses serviços para as aplicações na IoT. Em seguida são tratadas a *web* semântica e sua estruturação, o vocabulário semântico, ontologias, *semantic markup* (anotações semânticas), a computação pervasiva e autonômica.

2.1 *Internet* das Coisas

Segundo Atzori, et al (2010, p.2789), a IoT refere-se exclusivamente a objetos endereçáveis e suas representações virtuais em uma estrutura na *internet*, que terá um crescimento elevado em um cenário moderno das telecomunicações de rede sem fio. Alguns objetos como: sensores, telefones móveis, leitores de códigos de barra, por exemplo; podem conectar-se a outros ou transmitir os dados em tempo real por etiquetas RFID (*Radio-Frequency Identification*). O RFID é visto como um elemento essencial da IoT, por causa de sua capacidade de controlar um grande número de objetos exclusivamente identificáveis com o uso de códigos de produtos eletrônicos (SUN, 2012, p.110).

Segundo Perera, Zaslavsky e Georgakopoulos (2014, p.420), a IoT ao endereçar cada elemento de forma única, aumenta incrivelmente o número de elementos dentro de uma rede. Por esta visão, na *Internet* das coisas os objetos individuais da vida cotidiana, como carros, estradas, câmeras sem fios, cartazes inteligentes, geladeiras, ou até mesmo o gado pode ser equipado com sensores, que pode acompanhar informações úteis sobre esses objetos. Sendo assim, se os objetos forem exclusivamente endereçáveis e conectados à *Internet*, a informação sobre os mesmos pode fluir rapidamente através dos mesmos protocolos que conectam nossos computadores na *Internet* (BITTENCOURT, 2011, p.85).

Uma vez que estes objetos podem sentir o ambiente e se comunicar, eles tornam-se ferramentas para a compreensão da complexidade, e muitas vezes podem permitir respostas autonômicas a cenários desafiadores sem intervenção humana. Como também os conceitos de computação pervasiva (KILJANDER, et al., 2014, p.858) ou ubíqua relacionando-se com

a IoT, no sentido de que todos esses padrões são ativados por dispositivos com sensores de grande escala incorporados.

Para que a IoT tenha este avanço nas pesquisas com a diversidade de objetos a serem conectados, além dos conceitos das etiquetas RFID serem considerados umas das principais visões, conforme mostrado na Figura 1, existem ainda a construção do *Internet Protocol* (IP) para permitir que objetos inteligentes sejam conectados à *internet* utilizando toda uma identificação dos recursos da *web*, como uma interface REST (*Representational State Transfer*). A visão da *web* semântica que aborda as questões de gestão dos dados que surgem no âmbito da vasta quantidade de informações é trocada por objetos inteligentes, e os recursos estão disponíveis através da interface *web*. Esta visão é realmente importante na separação dos significados de dados e a ideia é que os significados semânticos dos objetos sejam armazenados separadamente a partir dos próprios dados e ferramentas eficazes para a gestão destas informações (AGGARWAL, et al., 2013, p.385).

Segundo Atzori (2010, p.2800), as visões da IoT podem ser sintetizadas a várias questões suscitadas e a Figura 1 é resultado de um trabalho de catalogação das principais áreas de pesquisa e tipos de aplicação oferecendo uma visão mais complexa da IoT. Uma melhor compreensão deste paradigma computacional, a IoT, é formado a parti da sobreposição de visões orientadas às coisas, à *internet* e à semântica. Mas este problema da sobreposição das visões é que estão incluídas as tecnologias.

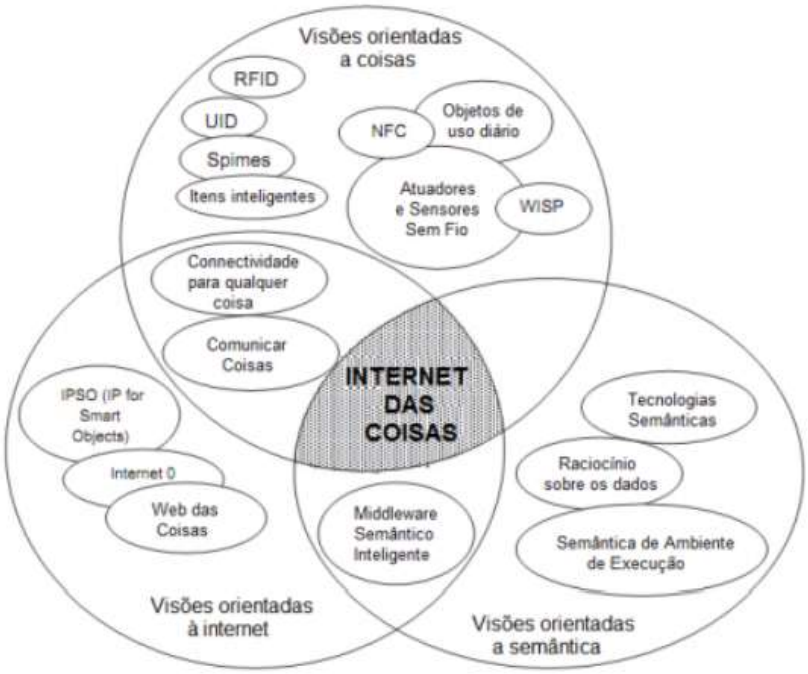

Figura 1 - Diagrama das diferentes visões na IoT. ATZORI, et al., 2010.

2.1.1 Serviços na IoT

Os serviços são abstrações adequadas para a construção de *software*s complexos e segundo Atzori (2010, p.2788) possuem papeis fundamentais para a IoT, fazendo parte de modelos de domínio e referência. É um termo que pode ter muitos significados, por exemplo, um uso difundido do termo serviço é usá-lo como um sinônimo para o que chamamos de uma interface técnica, ou funcionalidade do *software* fornecido por uma interface de serviço ou serviço *web*. Por Ferrario e Guarino (2009, p.90) algumas pesquisas foram realizadas para estabelecer uma definição única de serviço dentro da ciência dos serviços e serviços da *Internet*. Mas, nenhuma definição foi aceita pela comunidade e é questionável se alguma vez vai ser uma única definição, pois há muitos cenários diferentes e usos que devem ser considerados (JONES, 2005, p.88).

Segundo Bauer (2011, p.950), o princípio da IoT é extensão da *Internet* para o mundo físico, envolvendo uma interação com uma entidade física no ambiente. Esta entidade constitui as coisas na IoT que poderia ser um ser humano, animal, carro, loja ou um item de uma rede logística, aparelho eletrônico ou um ambiente fechado ou aberto, sendo o principal

foco das interações por seres humanos e/ou agentes de *software*. Esta interação é possível graças a um componente de *hardware*, um dispositivo ou sensor, que atribui a uma entidade ou é parte do ambiente de uma entidade para que ele possa monitorá-lo. O dispositivo permite que a entidade faça parte do mundo digital através da mediação das interações. E o recurso é o componente de *software* real que fornece informações sobre a entidade ou controla um dispositivo. Os serviços expõem funcionalidades de um dispositivo acessando seus recursos e com as implementações de recursos fornece uma interface bem definida e padronizada, para que as funções criadas interajam com entidades e processos relacionados. As relações entre os serviços e entidades são modeladas como associações. Estes conceitos identificados do domínio IoT e as relações podem ser representados como na Figura 2.

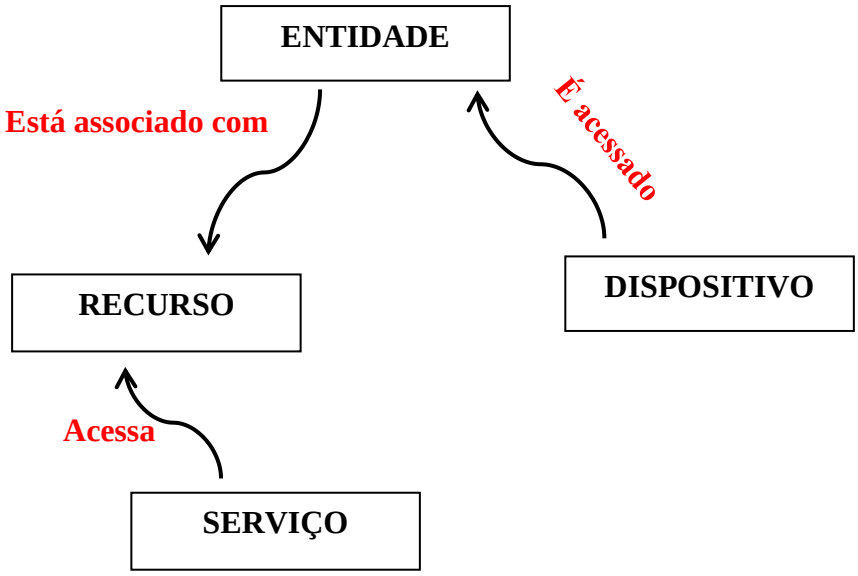

Figura 2 – Modelo da IoT: Interações e Conceitos Chaves. BAUER, et al., 2011.

Os conceitos identificados precisam ser modelados em um formato que forneça representações interpretáveis aos seres humanos e as máquinas interoperáveis. A comunidade *web* semântica introduziu definições formais especificadas como ontologias. A OWL(*Web Ontology Language*) e OWL-DL(*Web Ontology Language-Description Language*), fornecem um formalismo que corresponde a uma hierarquia de classes e inserir formalismos lógicos através dos axiomas para representar a informação (OLIVEIRA, et al.,

2009, p.5). Uma entidade pode ter certos aspectos que necessitam levar em consideração, como por exemplo, quando precisa saber sobre a localização de uma entidade ou as características de interesse para os dados disponíveis (BAUER, 2011, p.950).

Segundo Wang, et al (2012, p.1795), os serviços na IoT são uma subclasse dos serviços definidos na OWL-S (W3C, 2004, p.35). Portanto, um serviço da IoT pode ter um perfil de serviço e um processo que descrevem as suas propriedades funcionais e não funcionais (herdada da classe de serviço OWL-S). Esta relação da propriedade é definida entre um recurso na IoT e um serviço e seu tipo, como mostrado na Figura 3.

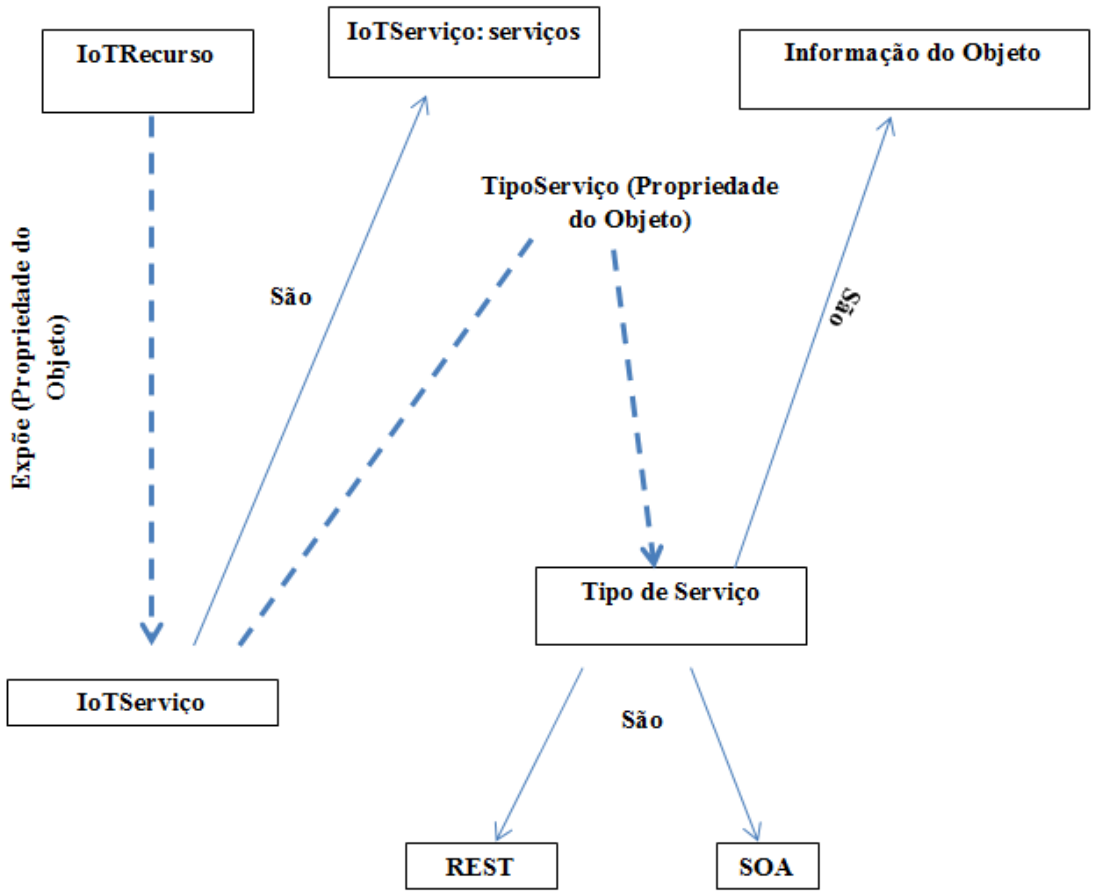

Figura 3 – Descrição da Ontologia de Serviço e Recurso na IoT. WANG, et al., 2012.

2.1.2 Estratégias de Descoberta

Na *web* semântica existem alguns formatos de anotações que são incorporados como estratégias para descoberta de serviços na *Internet* das coisas. Segundo Mayer e Guinard

(2012, p.121), a descoberta semântica de serviços para recursos *web* baseia-se em vários esquemas de mapeamento para identificar *semantic*amente estes recursos com endereços de rede conhecidos. Para fornecer esta funcionalidade utilizará qualquer uma das descrições de recurso: *Microformats*, *Microdata*, JSON (*JavaScript Object Notation*) ou RDF.

Um exemplo de simples da ligação entre dois dispositivos em conjunto via *ethernet* ou *wifi*, permitindo uma comunicação pelo protocolo IP, mas não garante que os seus serviços serão compatíveis. Para resolver este problema basicamente seria necessário instalar os *drivers*, por exemplo, ao conectar uma impressora e um computador na rede, pode ser necessário instalar o *driver* da impressora no computador. Os dois dispositivos citados com um dado protocolo de descoberta de serviço implementado, serão capazes de encontrar os serviços fornecidos por si e utilizá-los. O protocolo de descoberta SLP (*Service Location Protocol*) permite que os dispositivos possam descobrir serviços suportados por outros dispositivos (GUTTMAN, 1999, p.73).

Em algumas linguagens semânticas, como a OWL-S, para atender ao processo de descoberta foi utilizada a ferramenta *Matchmaker* configurada no repositório jUDDI (*Java Universal Description, Discovery, and Integration*) (OLIVEIRA, et al., 2009, p.7). As buscas são realizadas com base na passagem de parâmetros de entrada e saída, presentes no elemento do perfil, sendo descritos os serviços propostos.

Segundo Oliveira, Menegazzo e Claro (2009, p.8), na WSMO (*Web Service Modeling Ontology*), a descoberta pode ser realizada através da passagem do elemento *Goals* que o cliente deseja alcançar, podendo ter o acréscimo de uma ontologia ou do envio de parâmetros pertencentes às propriedades não funcionais do elemento *Web Service*. A utilização dos componentes se dá com a chamada do método *discover*(), que pode requisitar somente metas, ou com uma segunda opção onde seria necessário apresentar também os conceitos descritos na ontologia.

Para a linguagem SAWSDL (*Semantic Annotations for WSDL and XML Schema*), a ferramenta utilizada para descoberta dos serviços foi o *plugin Lumina*, juntamente com o repositório de serviços jUDDI. Esta relaciona os conceitos da ontologia com os parâmetros de entrada, de maneira que o sistema de busca possa consultar de acordo com a descrição realizada pelo modelo de referência.

2.2 *Web* Semântica

A *web* armazena uma enorme quantidade de informações criadas por diferentes organizações, comunidades e usuários que realizam diversos serviços como comércio, turismo, notícias, ensino ou movimentações bancárias. É interoperável apenas sintaticamente, ou seja, é projetada para processamento humano dos significados dos dados. Desta maneira, a *web* semântica é considerada uma ampliação da *web*, no sentido de prover uma infraestrutura aberta, com tecnologia de serviços *web*, para que haja compartilhamento e tratamento da informação e do conhecimento, entendido também por máquina (MOURA, 2003, p.70).

Algumas pesquisas iniciais da *web* semântica tem como foco a definição dos conceitos, tecnologias e padrões para possibilitar uma proposta semântica. Algumas linguagens de descrição semântica, como a WSDL (*Web Service Description Language*), OWL-S (*Ontology Web Language for Services*), WSMO (*Web Service Modeling Ontology*) e SAWSDL(*Semantic Annotations for* WSDL), são recomendadas pela W3C (*World Wide Web Consortium*) e têm sido estudadas para adicionar marcação semântica aos recursos da *web*. Entretanto, deve também ser levada em consideração a maneira de como a grande quantidade de aplicações que foram desenvolvidas durante anos na *web* atual continuarão a serem desenvolvidas na *web* semântica. Uma solução efetiva consiste em realizar acesso programático a essas aplicações e visualizá-las como objetos de primeira classe, que são elementos de uma linguagem que podem ser passados como parâmetros ou tratados como valores, podendo ser manipulados da mesma forma que dados (AGGARWAL , et al., 2013, p.390).

Para entender melhor os conceitos abordados dentro da *web* semântica, deve-se compreender que uma ontologia é um documento ou um arquivo no qual estão definidas formalmente as relações entre os conceitos. Assim sendo, é uma taxonomia formada de classes e subclasses de objetos relacionados entre si e com mais de um conjunto de regras de inferência que podem utilizar uma linguagem como a DAML (*Agente Mark-up Language)* desenvolvida como ontologia e inferência baseada em RDF, onde é um modelo de linguagem padrão para representar informações sobre recursos na *web*.

Esta arquitetura da *web* semântica está incorporada no paradigma da IoT, visto que todas as camadas devem ser implementadas para que as tecnologias possam a parti dai criar redes de "coisas inteligentes" que são encontradas no mundo físico, como tecnologias que tornam a IoT realizável, como a RFID (*Radio Frequency Identification*), WSN (*Wireless Sensor Network*) e WSAN (*Wireless Sensor and Actuator Network*).

Existe uma tendência em tratar a IoT em WoT (*web* das coisas), nos quais os padrões abertos da *web* são empregados para prover o compartilhamento de informação e a interoperabilidade entre dispositivos. Com efeito, a *internet* está a evoluir para a chamada "*Web of Things*" (WoT), um ambiente onde todos os dias objetos, tais como edifícios, calçadas, semáforos, e mercadorias são identificáveis, legível e reconhecível, endereçável, e até mesmo controlável através da *internet* pela a World Wide Web.

Alguns motivos favorecem a WoT, como: a tecnologia integradora dos serviços *web* que tem se mostrado indispensável na criação de aplicações distribuídas interoperáveis para *Internet*; coisas inteligentes (*smart things*) e com servidores *web* incorporados, que podem ser abstraídas como serviços *web* e perfeitamente integradas em dispositivos embarcados. Porém, é necessário observar que a semântica possui uma estrutura para que os vocabulários e ontologias sejam criados para que a comunicação entre as “coisas” seja realizada na IoT.

2.2.1. Estrutura Semântica

Para funcionar, a *web* semântica deve ter o acesso às coleções de informações estruturadas e aos conjuntos das regras de inferências que poderão ser usadas para conduzir a um raciocínio automático. As linguagens utilizadas para as regras deverão ser tão expressivas quanto possíveis para permitir que se possa raciocinar sobre a *web* tão amplamente quanto necessário, logo é vista como uma solução extraordinária para resolver problemas da *web* atual.

Analisando a arquitetura da *web* semântica, percebe-se que seus princípios são implementados em camadas de tecnologias e padrões *web*, conforme a Figura 4. As camadas Unicode e URI estabelecem um conjunto de caracteres internacionais e fornece meios para a identificação de objetos na *web* semântica. A camada XML, com definição de *namespace* e *schema*, estabelece o que se pode integrar nas definições da *web* semântica com outros

padrões baseados em XML. A camada RDF e RDF *Schema* possibilita estabelecer declarações sobre objetos com URI (*Uniform Resource Identifier*) e definir vocabulários que podem ser atribuídos por URIs. Esta é a camada onde se tem a capacidade de oferecer tipos para recursos e *links*. A camada da Ontologia suporta a evolução de vocabulários assim como pode definir relações entre conceitos diferentes. Já a camada da Assinatura Digital detecta as alterações em documentos. Esta é a camada que está sendo progressivamente padronizada por grupos de trabalho do W3C. As camadas da Lógica, da Prova e da Validação, estão sendo progressivamente pesquisadas e aplicações demonstrativas simples já estão sendo construídas. A camada da Lógica possibilita a escrita de regras. A camada da Prova executa estas regras e avalia, juntamente com a camada da Validação, os mecanismos que permitem às aplicações confiar ou não nas provas realizadas.

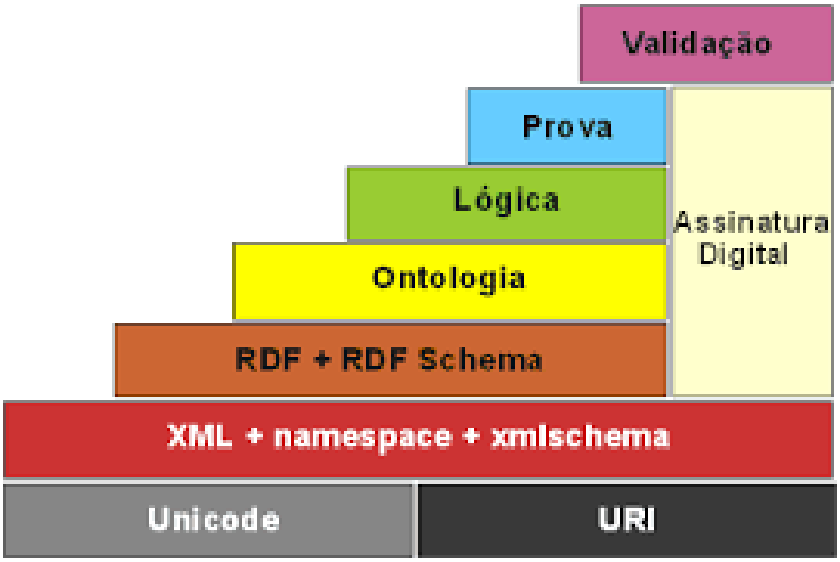

Figura 4 – Estrutura de camadas da *web* semântica. MOURA, 2003.

Os componentes da *web* semântica são:

Identificador – URI: (*Uniform Resource Identifier*), defini uma maneira padronizada de nomear recursos. Está ligado a algum tipo de arquivo publicado na *web*, como: foto; página, mas seu uso na *web* semântica é apenas identificar recursos.

Conjunto de caracteres – Unicode: padrão internacional para representação de caracteres fornece um único número para cada caractere.

Sintaxe – XML: (*Extensible Markup Language*) pode ser definida como uma linguagem de marcadores como a HTML e foi desenvolvida para escrever dados.

Intercâmbio de Dados – RDF: (*Resource Description Framework*) é uma linguagem que esta apta a prover um modelo de afirmações e citações em que se podem mapear os dados em qualquer novo formato.

Taxonomias – RDFs: (*Resource Description Framework Schema*) um *framework* que fornece meios para a criação de vocabulário as RDF, onde é possível descrever novas classes e propriedades e relacionamentos entre elas.

Consulta – SPARQL: (SPARQL *Protocoland RDF Query Language*) pode ser definido como o protocolo e linguagem para consultas a modelos RDF. Sua sintaxe é similar à linguagem SQL.

Ontologias – OWL: (*Web Ontology Language*) é uma extensão do RDF, que inclui termos de vocabulário para descrever classes, fatos e relacionamentos sobre estas classes e características destas relações atrás de axiomas lógicos.

Regras – SWRL: (*Semantic Web Rules Language*) são regras que visam novos conhecimentos em modelos RDF/OWL (MOURA, 2003, p.70).

2.2.2. Vocabulário Semântico

Na *web* semântica, vocabulários definem os conceitos e relações (também conhecidos como “termos”) utilizados para descrever e representar uma área do conhecimento. Então os vocabulários funcionarão como ferramentas conceituais que nos ajudam a descrever os dados e seus formatos. E são usados para classificar os termos que podem ser usados em uma aplicação particular, caracterizar possíveis relações e definir possíveis restrições sobre o uso desses termos.

Na prática, os vocabulários podem ser muito complexos (com vários milhares de termos) ou muito simples (descrevendo apenas um ou dois conceitos) (WHITEHEAD, 2006, p.72).

Pode-se entender que o papel dos vocabulários sobre a *web* semântica é para ajudar a integração de dados, quando existir algumas ambiguidades sobre os termos utilizados nos diferentes conjuntos de dados, ou quando um pouco mais de conhecimento pode levar à descoberta de novos relacionamentos. Considerando que a aplicação de ontologias no domínio na área de saúde, médicos podem utilizá-las para representar o conhecimento sobre os sintomas, doenças e tratamentos. As empresas farmacêuticas usam para representar suas informações sobre drogas, dosagens e alergias. Estas informações levam a combinações que permite aplicações inteligentes, como ferramentas de apoio à decisão, que buscam por possíveis tratamentos e que os sistemas que monitoram a eficácia dos medicamentos averiguem possíveis efeitos colaterais.

Outro tipo de exemplo é a utilização de vocabulário para organizar o conhecimento. As bibliotecas, museus, jornais, portais governamentais, empresas, aplicações de redes sociais, e outras comunidades que gerenciam grandes coleções de livros, artefatos históricos, reportagens, glossários de visita, entradas de *blog* e outros itens podem agora usar vocabulários, usando formalismos padrão, a alavancar o poder de dados vinculados (HACHEM, et al., 2011).

A depender da aplicação alguns aplicativos podem decidir não usar, estes pequenos vocabulários, e contando somente com a lógica do programa de aplicação. Outros podem optar por utilizar vocabulários muito simples e deixar um ambiente da *web* semântica utilizar essa informação extra para fazer a identificação dos termos. E outros precisam de um acordo sobre terminologias comuns, sem qualquer rigor imposta por um sistema de lógica. Finalmente, alguns aplicativos podem precisar de ontologias mais complexas com procedimentos de raciocínio complexo, que dependerá das necessidades e os objetivos das aplicações.

Segundo Kalaoja, et al (2006, p.465), para permitir uma representação rica de serviços e facilitar a descoberta eficiente de serviço e composição, capacidades funcionais, entradas, saídas e não-funcionais dos atributos dos serviços pode ainda semanticamente serem anotados utilizando vocabulário de ontologias externas. O uso de ontologias permitirá que as entidades e serviços computacionais possam ser mais claros e permitem um melhor raciocínio em suas propriedades. O sistema Amigo proposto por Kalaoja, et al (2006, p.468)

descreve um conjunto comum de conceitos e vocabulários desenvolvidos para o ambiente doméstico em rede, onde qualquer casa pode ser consideravelmente facilitada pela sua utilização.

Um exemplo de vocabulário pode ser observado pelo grupo incubador geoespacial, privilegiados por iniciar questões de localização e propriedades geográficas dos recursos para a *web* de hoje e de amanhã. Um passo importante realizado pelo grupo foi para atualizar o vocabulário W3C GEO, lançando as bases para uma ontologia geoespacial mais abrangente, formulando uma proposta para desenvolver recomendações a promover a representação da *web* da localização física e geografia (LIEBERMAN, 2007, p.8).

O trabalho da Incubadora tem sido grandemente influenciado pelo trabalho do Geoespacial Aberto Consortium (OGC), onde é essencial para a clareza das representações espaciais, como a amplitude e a profundidade da informação geográfica, promovendo o desenvolvimento de vocabulários geoespaciais e considerando os aspectos geoespaciais de outras atividades do W3C.

2.2.3. Ontologias

O termo ontologia possui origem no grego ontos, ser, e logos, palavra. Ontologia é um catálogo de tipos de coisas em que se supõe existir um domínio, na perspectiva de uma pessoa que utiliza uma determinada linguagem. Uma das definições mais conhecidas para ontologias é apresentada como:

> Uma ontologia é uma especificação explícita de uma conceitualização. [...] Em tal ontologia, definições associam nomes de entidades no universo do discurso (por exemplo, classes, relações, funções, etc. com textos que descrevem o que os nomes significam e os axiomas formais que restringem a interpretação e o uso desses termos) [...]. Ou seja, ontologia é uma descrição (como uma especificação formal de um programa) dos conceitos e relacionamentos que podem existir para um agente ou comunidade de agentes (GRUBER, 2009, p. 210).

Na computação em geral o termo ontologia foi introduzida no início da década de 90, onde tem sido bastante utilizado, englobando um conjunto de definições de conceitos, propriedades, relações, restrições, axiomas, processos e eventos que descrevem certo

domínio ou universo de discurso. Essas definições habilitam aplicações e agentes de *software* a utilizarem semântica formal, precisa e clara para processar a informação descrita pela ontologia e usar essa informação em aplicações inteligentes (PINHEIRO, 2009, p.35).

As ontologias são tratadas como um artefato computacional composto de um vocabulário de conceitos, definições e suas possíveis propriedades, representando, de maneira clara e não ambígua, o conhecimento do domínio. São definidas em dois tipos: ontologia de domínio ou vertical e ontologia de nível superior, independente de domínio, ou ontologia horizontal. Ontologias de domínio definem conceitos para um domínio específico, por exemplo, zoologia, medicina, matemática. Ontologias de nível superior são aquelas em que os conceitos independem do domínio, ou seja, podem ser utilizadas em diversos domínios. Alguns exemplos de ontologias de nível superior ou horizontal são ontologias para serviços *web*, ontologia de tempo, dentre outras (FENSEL, et al., 2011, p.88).

Os componentes básicos de uma ontologia são: classes que expressam qualquer coisa sobre a qual alguma coisa é dita e são organizadas em uma taxonomia; relações que representam o tipo de interação entre os conceitos de um domínio; axiomas, utilizados para modelar sentenças sempre verdadeiras; e instâncias: utilizadas para representar elementos específicos, ou seja, os próprios dados (HACHEM, 2011, p.3). A tarefa de se criar uma ontologia para um domínio específico não é uma atividade trivial. Dessa forma, antes de partir para a criação de ontologias deve-se ter em mente a real necessidade de se ter a ontologia. Para criar uma ontologia podem ser utilizadas algumas ferramentas como, por exemplo:

- ✓ Protégé 2000: um ambiente interativo para construção de ontologias, que oferece uma interface gráfica para sua edição, Figura 5. Arquitetura modulada permitindo inserção de novos recursos. Desenvolvido pelo grupo de informática médica da Universidade de Stanford (http://www.stanford.edu/), possuindo código aberto em uma aplicação *standalone*, composta por um editor de ontologia e uma biblioteca de *plugins* com funcionalidades. Atualmente importa/exporta para diversas linguagens, dentre elas Flogic, OIL (*Ontology Inference Layer*), XML e Prolog (KNUBLAUCH, et al., 2004, p.231).

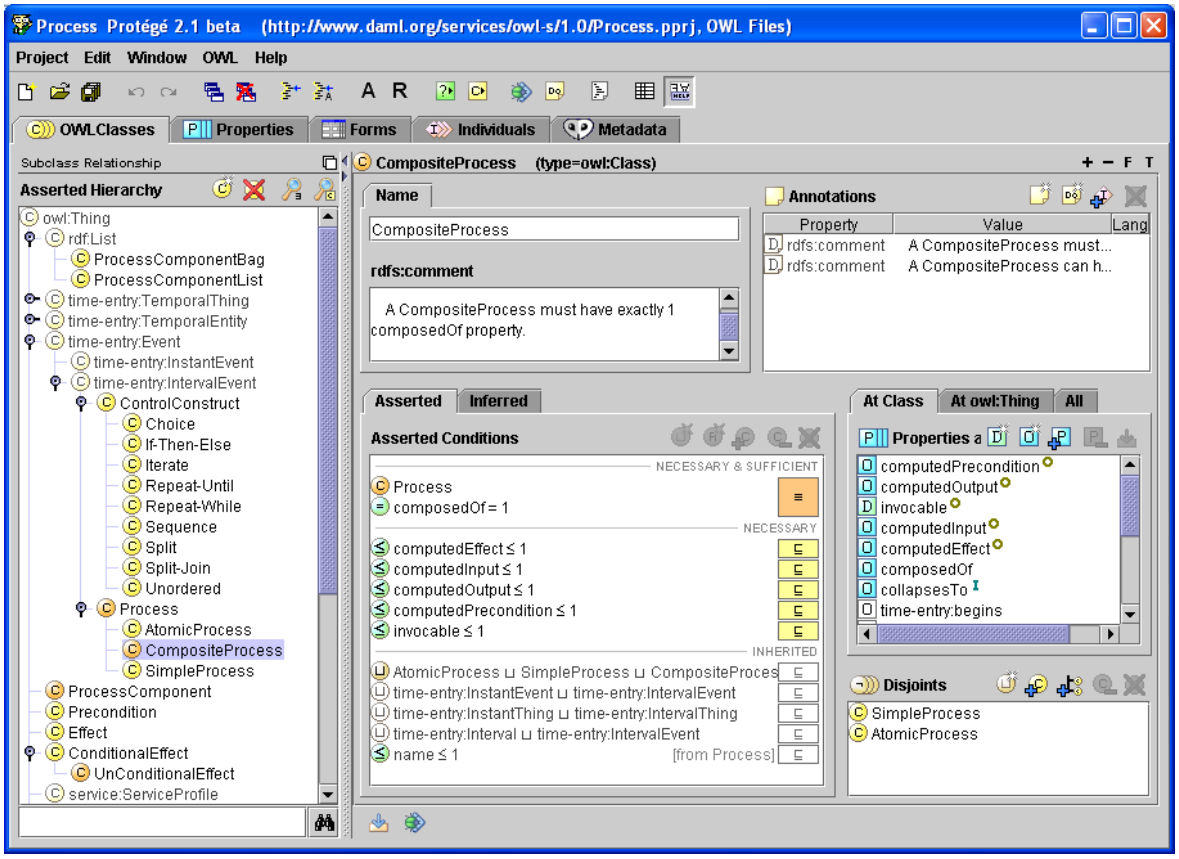

Figura 5 – Protégé 2000. KNUBLAUCH, et al., 2004.

- ✓ OntoEdit: ambiente gráfico para edição de ontologias que permite inspeção, navegação, codificação e alteração de ontologias. Desenvolvido pela AIFB (*Institut für Angewandte Informatik und Formale Beschreibungsverfahren)* na Universidade de *Karlsruhe*, possui arquitetura extensível baseada em *plugins* e atualmente está disponível nas seguintes versões: *OntoEdit Free* e *OntoEdit Professional*, conforme Figura 6 (SURE, et al., 2002, p.221).

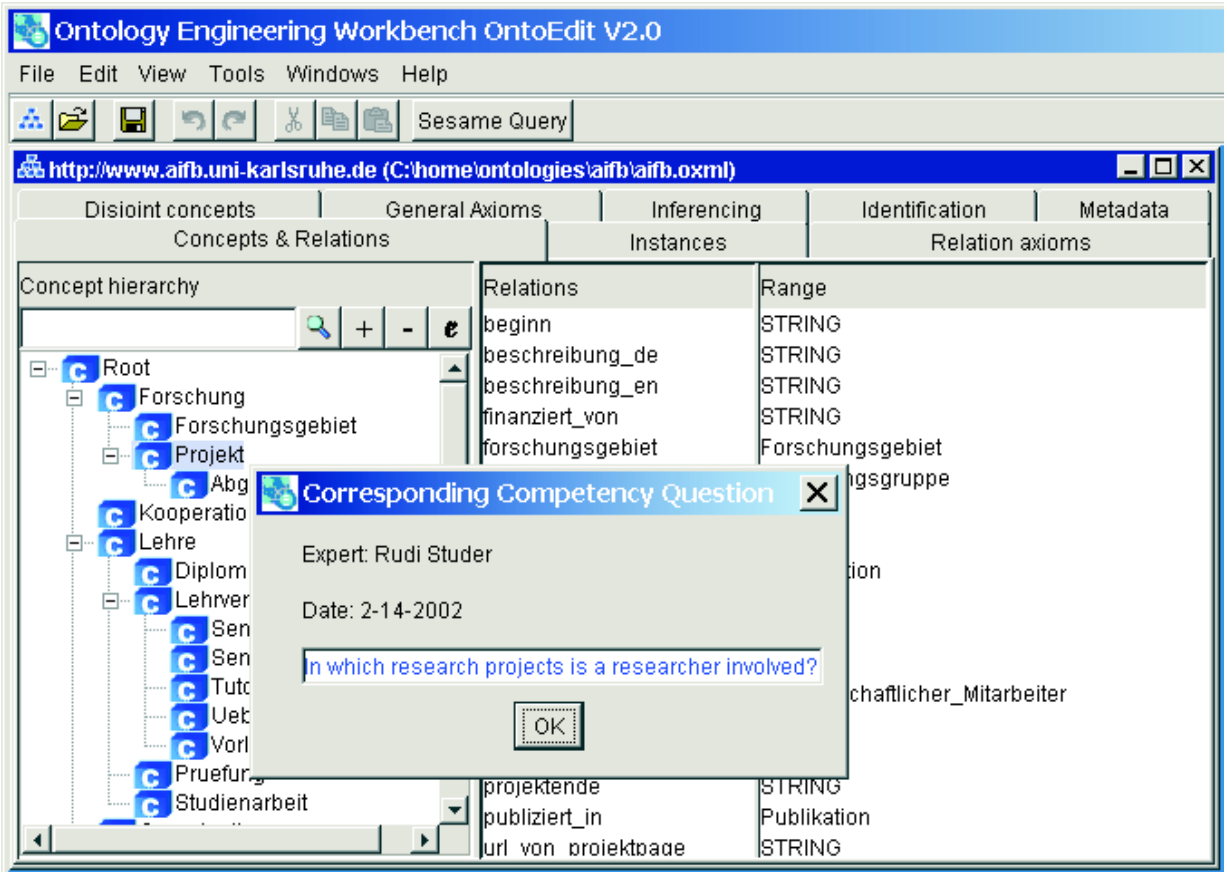

Figura 6 – Ontoedit. SURE, et al., 2002.

2.3. Anotação Semântica (*Semantic Markup*)

Algumas maneiras para definição de conteúdo semântico têm surgido desde a década de 90 pela W3C, como os padrões de marcação semânticos: RDF, RDFa, *Microformats* e *Microdata*. O processo para incluir informação semântica ou metadados é chamado de anotação, e significa adicionar informações legíveis por máquina para o conteúdo existente. Desta maneira, a tecnologia pode suportar uma multiplicidade de aplicações ou descobrir mais precisamente o conteúdo.

Um modelo abstrato é utilizado pelo RDF, onde decompõe a informação em pedaços pequenos, utilizando regras simples sobre a semântica. O objetivo é proporcionar um método geral que é simples e flexível o suficiente para expressar qualquer fato. Este modelo abstrato tem os seguintes componentes principais: declaração (formalmente chamado como um triplo); sujeito e objeto recursos; e predicado. Onde o sujeito e objeto são nomes de duas

coisas no mundo, e o predicado sendo o nome que relaciona estas duas coisas, conectando-as (NCE, 2011).

Comparando com a marcação, para acelerar a busca e ajudar a encontrar informações relevantes e precisas, a anotação semântica entra em um nível mais profundo, onde pode enriquecer os dados não estruturados ou semiestruturados com um contexto que é mais ligada ao conhecimento estruturado de um domínio e permite que os resultados não estejam explicitamente relacionados à busca inicial. Assim, se a marcação estiver pronta a encontrar o resultado mais relevante, anotação semântica acrescentará diversidade e riqueza para o processo.

A anotação semântica é, essencialmente, uma forma significativa para descrever a estrutura e aparência de um documento em particular. Ajuda na junção de ambiguidades entre a língua natural e sua representação computacional em uma linguagem formal. Então o processo de marcação, deve criar primeiramente uma coleção de declarações RDF para descrever o significado de um documento da *web*, em seguida, colocá-los em um arquivo separado e, finalmente, ligar de alguma forma o documento original da *web* para este arquivo RDF. Outra maneira mais simples de realizar estas operações seria a utilização dos *Microformats* desenvolvido por Tantek Celik, Chris Messina, entre outros (WHITEHEAD, 2006, p.70).

A principal vantagem da *Semantic Markup* é melhorar as buscas das informações do outro lado da *Internet* e das bases de dados úteis aos usuários. Alguns exemplos podem ser vistos nos principais sites de buscas como: *Bing*, *Google*, *Yahoo*.

2.3.1. MICROFORMATS

Os *Microformats* não são mais uma nova linguagem e sim uma ideia para resolver os problemas mais simples primeiramente da *web* semântica. A sua adaptação aos padrões atuais e utilização na marcação semântica é de fácil utilização. A proposta dos *microformats* é bem simples, onde os formatos mais usados, estão o *hCalendar* (para publicação de eventos) e *hCard* (para pessoas, empresas e organizações em geral).

Os *microformats* são coleções de formatos para embutir metadados de documentos no interior de suas páginas reutilizando atributos existentes em HTML como, por exemplo,

class e *title*. Um exemplo de aplicação recorrente de microformatos é na inserção dos metadados em páginas de agenda de eventos e contatos pessoais. São utilizados com vocabulários próprios, herdados de vocabulários já existentes e com grande utilização permitindo a inserção de dados semânticos em páginas HTML de forma rápida e fácil por reutilizar atributos e elementos próprios do HTML e que não irão causar danos para validações de páginas junto ao W3C (WHITEHEAD, 2006, p.71).

Uma abordagem que inclui os conceitos de RDF e a ideia dos *Microformats* descrevem uma solução ideal para as páginas HTML, chamado de RDFa (RDF – *in* – *attributes*). Os mesmos podem ser diretamente incorporados XHTML para transmitir o significado do documento em si (TOMBERG, et al., 2013, p.103).

2.3.2. RDFS (*Resource Description Framework*)

Linguagem baseada em XML, criada pela W3C em 1999, foi concebida com o intuito de representar informação na *Internet*. É um modelo similar aos conceitos de modelagem ER (entidade-relacionamento). Sendo considerada uma linguagem abrangente, é possível definir ontologias com ela. Mas, por ser uma linguagem complexa se torna hoje inviável utilizá-la para marcar páginas *web semantic*amente. Conforme foi tratado anteriormente o RDFa (RDF – *in* – *attributes*) é o padrão de anotação semântico que não possui a necessidade de utilização de novos elementos de marcação, sendo realizado por atributos incorporados em qualquer elemento (NCE, 2011).

A diferença do RDF para RDFa é que primeiramente a marcação era no próprio conteúdo, mas no RDFa criou-se um conjunto de extensões de XHTML obedecendo aos padrões XML, sendo obrigatório utilizar de *namespaces* para todo e qualquer escopo/atributo de item semântico que fosse necessário descrever, tornando assim a marcação semântica trabalhosa. Além disso, foi projetado para fornecer interoperabilidade e semântica para metadados visando facilitar a busca na *web*. RDF é um padrão de metadados recomendado pelo W3C como linguagem para recursos interligados, utilizando de 03(três) princípios fundamentais: recursos (podem ser pessoas, lugares eventos, etc.); propriedades; e frases.

2.3.3. MICRODADOS

Os microdados são considerados o mais recente concorrente entre os padrões de anotações da *web* semântica e criado também pela W3C como parte do HTML5. Fortemente influenciado pelos microformatos, adotando uma série de atributos como *hCard, hCalendar* e outros microformatos. Além dos elementos semânticos, o HTML5 introduz os microdados de maneira que as anotações das páginas *web* com metadados semânticos utilizem atributos *javascript,* ao invés de documentos XML separados. Qualquer um pode definir um vocabulário microdados e começar a incorporar propriedades personalizadas em suas próprias páginas *web*. Cada vocabulário microdados define um conjunto de propriedades nomeadas (NCE, 2011).

Um exemplo seria um vocabulário onde a pessoa poderia definir imóveis como nome e foto. Para incluir uma propriedade microdados específico na sua página *web,* você fornece o nome da propriedade em um lugar específico. Dependendo de onde você declarar o nome da propriedade, os microdados possuem regras de como extrair o valor da propriedade.

Os primeiros vocabulários microdados foram definidos em 2010 e podem ser encontrados na *Data Vocabulary.org*, onde existem poucos vocabulários definidos: *Event, Organization,Person, Product, Review, Reviewaggregate, Breadcrumb, O ffer* e *Offer-aggregate*. Segundo a NCE (2011), o melhor microdados é um repositório de vocabulários chamado de *Schema.org*.

Existe uma lista em torno de 450 formatos definidos no *schema.org* que se tornou um dos principais motores de busca para melhor indexar, classificar e refinar seus resultados de busca, visto que o *schema.org* é na verdade um consórcio entre os grandes buscadores da *web – Google*, *Bing* e *Yahoo*! Existe um incentivo à utilização de microdados, devido a esta lista de formatos e existe uma razão para usar o RDFa em vez do microdado. A simplicidade consegue fazer o microdado dar maior resultado e faz parte do HTML5.

2.4 Arquitetura REST (Representation State Transfer)

Segundo Zeng (2011, p.426), a arquitetura baseada em REST é considerada “a verdadeira arquitetura da WEB”, onde seu conceito é modelado como recurso HTTP usando a URI. Os clientes podem identificar os recursos de que necessitam através da URI, passando

a manipular os mesmos através dos comandos tradicionais do HTTP, como: *PUT, GET, POST e DELETE*. O *PUT* e o *DELETE*. Estes comandos podem ser empregados para a criação e deleção dos recursos e *GET* e *POST* para recuperar e alterar os recursos.

Esta arquitetura possui mensagens autodescritivas, ou seja, os recursos são livres para realizar suas próprias representações de formato de dados. Obviamente, os sistemas finais precisam concordar com essa representação para que a comunicação possa ocorrer de forma adequada. Outra característica importante é que o REST opera com requisições *stateless*, tratando cada requisição de forma independente, podendo com isso não necessitar de um servidor para armazenar informações de sessão ou sobre o *status* de como está cada uma das múltiplas requisições (ZENG, 2011, p.430).

Vale ressaltar que o REST não é limitado somente ao protocolo HTTP, isto é qualquer protocolo da camada de aplicação que forneça um vocabulário rico e padronizado pode ser utilizado para a transferência de dados. Dessa forma, qualquer protocolo de rede pode ser aplicado sem grande esforço.

2.5 Computação Pervasiva

Atualmente as redes sociais fazem parte do dia a dia de diversos usuários da *Internet*. As informações que podem ser obtidas em um estudo detalhado do perfil de um usuário podem ser reveladoras: desde traços de seu comportamento, que podem ser usados para verificação em uma seleção para um emprego, como interesses pessoais, que podem ser úteis para outros usuários descobrirem novos amigos ou novas informações sobre determinado assunto.

Com a popularização e crescente difusão de dispositivos de comunicação móveis apresentam maior capacidade computacional. Um novo cenário se apresenta, onde as informações disponíveis nas redes sociais podem ser associadas com informações de localização, e tudo isso fica disponível a partir dos próprios dispositivos móveis. Assim, as Redes Sociais Pervasivas (RSP) representam um novo paradigma de computação derivado da convergência da Computação Pervasiva com os Serviços de Redes Sociais da *web* 2.0, como por exemplo, *Facebook, Orkut*, *MySpac*e, *Twitter*).

Na visão tradicional de computação pervasiva, usuários de dispositivos móveis descobrem e interagem com serviços oferecidos pelo ambiente físico em seu entorno. Já com as RSP os usuários podem também descobrir pessoas dentre suas relações sociais com interesses similares. Diferente de serviços de *web* 2.0 que conectam pessoas que são relacionadas através da exploração de suas relações sociais, RSP tem como foco a descoberta de pessoas que estão ao mesmo tempo socialmente e fisicamente próximas (MASUOKA, 2003, p.69).

Em uma aplicação típica de RSP, usuários descrevem seus interesses na realização de atividades em conjunto com suas preferências sociais, indicando com quem, dentre seus indivíduos de elo social direto, preferem realizar suas atividades. Passam essa informação para a aplicação de RSP hospedada em seu dispositivo móvel (que pode ser um *smartphone* como, por exemplo, *iphone*, *blackberry* ou dispositivo móvel com a plataforma *android*), e esperam em resposta recomendações de usuários com interesses similares, com quem eles estão direta ou indiretamente conectados na rede social e de quem eles estão próximos geograficamente.

2.6 Computação Autonômica

A computação autonômica surgiu a partir de uma solução proposta para resolver o problema da excessiva complexidade dos *software*s corporativos. A ideia basicamente segue os conceitos e caraterísticas dentro das diversas ramificações da engenharia, buscando inspiração no sistema nervoso autonômico humano, mantendo as funções vitais, isto é em pleno funcionamento, a fim de compreender e resolver os diversos tipos de problemas sem qualquer intervenção direta do ser humano.

Então, podemos definir que um sistema autonômico é capaz de se gerenciar de acordo com objetivos de alto nível definidos pelo administrador. A principal meta deste sistema é a capacidade de autogerenciamento, livrando o administrador de sistemas da preocupação com detalhes operacionais e possibilitando ao usuário empregar as máquinas com melhor desempenho durante todo o tempo.

Juntamente com o objetivo da computação ubíqua (STERRITT, et al., 2003, p.249) que propõe possibilitar ao usuário o acesso a ambientes computacionais, em qualquer lugar e

a qualquer momento, surge um paradigma computacional com possível capacidade de auxiliar de forma menos intrusiva os seres humanos em suas atividades profissionais.

Para atingir os objetivos mencionados e ser considerado um sistema de computação autonômica é necessário atender a quatro serviços básicos de acordo com Parashar e Hariri, (2005, p.20) e conforme a Figura 7: auto-configuração (*Self-Configuring*), capacidade do sistema de se configurar em tempo de execução; auto-otimização (*Self-Optimizing*), capacidade do sistema em otimizar a alocação dos recursos existentes; auto-cura (*Self-Healing*), habilidade do sistema de identificar falhas e executar ações corretivas, sem que para isso seja necessário paralisar o funcionamento dos serviços; auto-proteção (*Self-Protecting*), habilidade do sistema em detectar comportamentos maliciosos ou hostis e tomar decisões eficientes e eficazes de forma transparente sobre qual é a ação mais adequada para impedir ou minimizar um ataque.

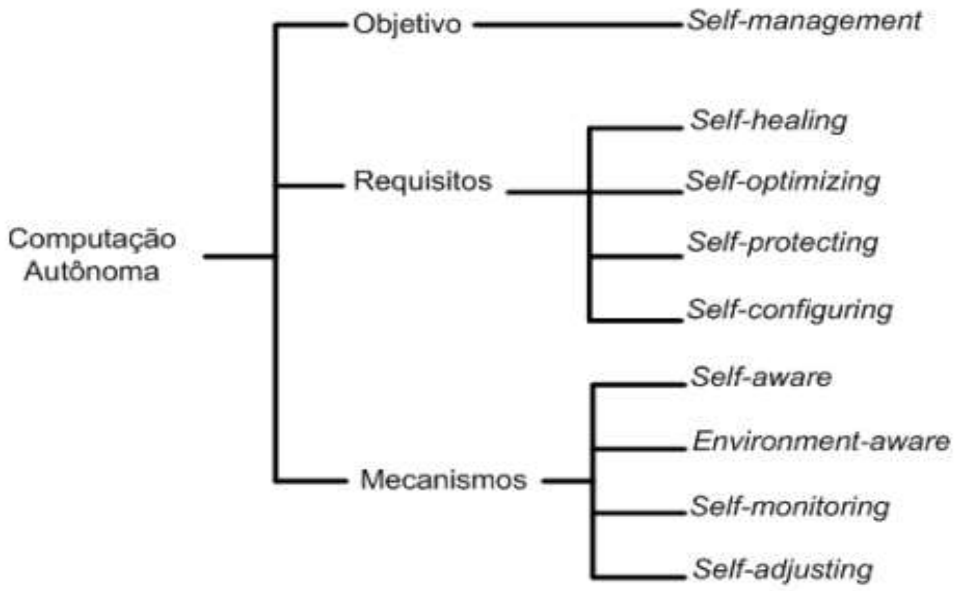

Figura 7 – Propriedades gerais para computação autônoma. PARASHAR, 2005.

3 TRABALHOS CORRELATOS

A criação de um vocabulário é imprescindível para definir as regras na *web* semântica e durante a pesquisa bibliográfica foram encontrados alguns trabalhos que construiram vocabulários próprios para uma determina aplicação. Para melhor entendimento da utilização de um vocabulário foram correlacionados os trabalhos de Kalaoja (2006, p.464), Villamor et al., (2010, p.5), Lanthaler e Gutl (2013, p.4), e Lieberman et al., (2007, p.6).

3.1 Vocabulário para serviços residenciais

Kalaoja, et al (2006, p.463) elaboraram um vocabulário para a modelagem semântica do *Home Services*, um sistema para casas que mostra como pode ser visualizado o manuseio de um número de aspectos ou domínios de aplicações para casas inteligentes, demonstrando várias propriedades e funcionalidades suportadas pelas pessoas que vivem na casa. Algumas dessas funcionalidades poderiam ser: reprodução de músicas da lista de preferencial; mostrar as notícias personalizadas; os resumos gerais dos tópicos das notícias; selecionar as configurações corretas para eletrodomésticos; ou baixar receitas e programas de culinária para a cozinha e exibi-los para facilitar a preparação dos alimentos.

Para verificar e validar esta abordagem reuniram requisitos de vários domínios de ontologias, como por exemplo, a domótica, que está relacionada com os requisitos dos dispositivos para automação residencial. Além disso, algumas classificações de ontologias foram consideradas como, a UPnP (*Universal Plug and Play*), fornecendo informações adequadas dos recursos do modelo do dispositivo. Alguns projetos de vocabulários para diferentes domínios com base em várias fontes foram analisados e classificados, descrevendo taxonomias preliminares e hierarquias de classes para uma ontologia.

A taxonomia proposta é baseada em várias descrições das propriedades dos dispositivos. Cada classe nesta taxonomia é uma especialização de suas classes pai. Para melhor esclarecimento, a taxonomia é restrita a três níveis, onde o primeiro define um papel da plataforma, o segundo uma subclassificação de plataformas fornecidas e o terceiro nível é geralmente uma tecnologia específica possivelmente com várias instâncias. O vocabulário

pode descrever, por exemplo, a propriedade de comunicação de uma plataforma móvel, ou classes de dispositivos móveis com propriedades típicas da plataforma.

Os autores apresentam a análise de requisitos e os resultados iniciais sobre classificações de vocabulários e ontologias na modelagem dos serviços e capacidades dos dispositivos em alguns domínios funcionais como, celular, computador pessoal ou domótica, que estão presente no ambiente da casa.

3.2 Vocabulário para serviços de imagens

Villamor, et al (2010, p.7) definiram um vocabulário para descrever serviços que realizam operações de recuperação, ou seja, serviços de pesquisa. A tarefa de modelagem foi restrita em considerar somente os serviços de pesquisa de imagem que tem o objetivo de encontrar um determinado tipo de imagem para o usuário.

Para executar esta tarefa, os autores seguiram os seguintes passos: identificação dos serviços de pesquisa de imagem populares; recolher os recursos através dos serviços considerados; modelar através de definições dos recursos; e adaptar os serviços para ajustar a interface modelada. A partir disso foi implementado um estudo de caso para definir o vocabulário, chamado de *MetaSearch Service*, onde ele considera uma descrição do *Microservice* analisando as combinações de definições dos recursos, construindo a interface para o usuário, permitindo executar o serviço.

Após os autores analisarem a semântica da descrição dos serviços e discutirem como o acesso aos dados poderia ser melhorado por meio da automação da sua execução, utilizaram um quadro de *Microservice* para construir um vocabulário que fosse empregado na construção semântica das descrições dos serviços de pesquisa de imagem na *web*. A solução foi um pequeno conjunto de termos que ilustra a abordagem orientada a funcionalidade do quadro e destaca a versatilidade de semântica leve. O vocabulário tem sido utilizado para descrever *s*emanticamente um conjunto de serviços que são agregados em um *MetaSearch* como estudo de caso. Resultados preliminares foram demonstrados no estudo de caso, onde o controle de interface do usuário é mostrado para cada opção de pesquisa, com base na semântica de cada parâmetro de entrada, e onde conjuntos de recursos

adequados são selecionados de acordo com os dados enviados (VILLAMOR, et al., 2010, p.7).

3.3 Vocabulário para APIs *web*

Lanthaler e Gutl (2013) criaram o Hydra, um vocabulário leve para descrever APIs *web* e para aumentar dados vinculados com controles de hipermídia. Com isso, permitem aos desenvolvedores alavancar na expressividade do RDF (*Resource Desription Framework*) com os benefícios do RESTs nos termos de escalabilidade e na criação de APIs interoperáveis que podem ser acessados por clientes genéricos.

As APIs têm tipicamente um modelo de dados bem definido e a documentação normalmente define uma série de objetos JSON, classes com suas propriedades. As propriedades são marcadas como opcionais ou obrigatórios, especificando o seu valor, e normalmente define se são somente leitura, somente escrita, ou leitura e escrita. Para transformar em JSON clássico as APIs *web* em uma API *Linked Data* está identificada as classes e propriedades com as IRIS (*Internationalized Resource Identifier*), complemento do *Uniform Resource Identifier* (URI) e uma sequência de caracteres do *Unicode* contidas no RDF.

De acordo com os autores os modelos de dados são usados para descrever uma informação específica em um domínio da aplicação e os vocabulários são usados para definir conceitos que podem ser compartilhados entre vários domínios de aplicação. Então estes modelos são normalmente utilizados para especificar os critérios de validade e restrições para os dados processados dentro de um aplicativo enquanto vocabulários são usados para a razão sobre os dados para descobrir novos conhecimentos. Para simplificar a integração de dados e permitir a reutilização, ele seria, assim, sensível para descrever os dados e comportamentos expostos por uma API *web* usando vocabulários RDF (LANTHALER , et al., 2013).

Desta maneira, foi projetado Hydra, um pequeno vocabulário estendendo o RDF *Schema* com conceitos necessários para descrever APIs *web*. A ideia básica é proporcionar um vocabulário que permita um servidor anunciar transições de estado válido para um cliente. Um cliente pode então usar esta informação para construir solicitações HTTP que

modificam o estado do servidor para que um determinado objetivo desejado seja alcançado. Desde todas as informações sobre as transições de estados válidos são trocados de forma processável pela máquina em tempo de execução em vez de serem codificados para o cliente no momento da concepção, os clientes podem ser dissociados do servidor e se adaptar às mudanças mais facilmente. Assim, permitem interagir com uma API *web* além de solicitações GET simples contendo a noção das operações solicitadas pelo HTTP. Além disso, o Hydra tenta preencher a lacuna que existe entre a combinação do estilo REST e dos princípios *Linked Data* que oferecem oportunidades para o avanço da *web*.

Os autores concluem que o Hydra foi projetado para ser um vocabulário modular, onde vocabulários futuros possam ser facilmente criados para estender a expressividade do Hydra. As pesquisas futuras seriam de investigar como a disponibilidade da informação do equipamento processável pode ser utilizada no processo de desenvolvimento, uma vez que a funcionalidade pode ser descrita antes de ser aplicada.

3.4 Vocabulário para localização de recursos *web*

Lieberman, Singh e Goad (2007, p.5), definem uma ontologia básica e vocabulário OWL para a representação de propriedades geoespaciais para recursos da *web*. É discutida a necessidade de atualizar o vocabulário Geo W3C (2003, p.25), apresentando um modelo para propriedades de recursos básicos da *web*. Apresenta realizações das propriedades dos elementos como XML e vocabulários OWL / RDF e descreve o uso dos elementos XML em *web feeds*.

O grupo incubador geoespacial da W3C, questiona a localização e propriedades dos recursos para a *web* de hoje e do futuro atualizando o vocabulário geo W3C, incluindo bases de ontologias mais compreensivas e promovendo uma representação da localização física e geográfica da *web*.

Os autores concluem que este modelo proporciona uma propriedade geral característica que pode ser usada para caracterizar qualquer conteúdo apropriado com uma característica geográfica. Propriedades específicas, tais como <*where*> associadas ao recurso com um número limitado de tipos de geometria que fornecem uma representação numérica para análise e visualização. Outras subpropriedades descrevem os atributos dos recursos

usados adicionalmente, como o nome do recurso e tipo do recurso (LIEBERMAN, et al., 2007, p.6).

3.5 Comparação com os trabalhos correlatos

Existem algumas diferenças entre os trabalhos aqui analisados e o presente trabalho. Primeiramente, todos utilizam um vocabulário especifico para um dispositivo ou aplicação. Alguns incorporam mecanismos de busca para serviços e não existe um padrão nas implementações do vocabulário.

Em relação ao trabalho proposto apresenta algumas vantagens como de poder simular a execução de um vocabulário semântico padrão, ou parte dele, para descobrir qualquer serviço *web* que esteja semanticamente definido para dispositivos conectados na IoT.

Na Tabela 1 é realizada uma comparação entre os trabalhos correlacionados neste capítulo.

Tabela 1 – Comparação entre os trabalhos correlatos.

Característica	Home Services	MicroServices	HiperMedia	Geospatial
Criação de Vocabulário	X	X	X	X
Análise de similaridade semântica	X	X	X	X
Classificação de serviços	X	X	-	-
Simulação do vocabulário	X	X	X	-
Testes com dispositivos da IoT	X	-	-	-

Fonte: Elaborado pelo autor, 2016.

Dessa forma, esta dissertação introduz uma forma de tentar preencher as lacunas na Tabela 1, visando a autodescoberta dos serviços da IoT, tentando tornar a comunicação dos dispositivos mais eficientes e eficazes como será explicado no capítulo 4.

4 PROJETO DO VOCABULÁRIO SEMÂNTICO

Para alcançar o vocabulário semântico, foram utilizadas as definições de serviços conforme a arquitetura orientada a serviços ou SOA (*Service-Oriented Architecture*), onde um serviço é uma função de um sistema computacional disponibilizado para outro sistema, possuindo uma *interface* definida. No escopo deste capítulo é definido o vocabulário semântico que realizar o descobrimento de serviços utilizados pela IoT. O vocabulário criado é demonstrado através de uma relação com seus objetos e definidos em seguida por classes e propriedades.

A validação do vocabulário criado foi baseada na arquitetura REST *Web Services*, como pode ser visto no capítulo 2, através de um mecanismo dinâmico de descobrimento de serviços de alto nível para IoT. A arquitetura REST *Web Services* tem como protocolo de aplicação o HTTP, um protocolo genérico que pode ser usado para várias tarefas, como serviço de nomes e sistemas de gerenciamento de objeto distribuído. Baseado nos seus métodos de requisições, conhecidos como requisições HTTP é possível construir sistemas, independente dos dados sendo transferidos, além de definir oito métodos: GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS e CONNECT (LEMOS, 2014).

Um exemplo de aplicação que utilizam os métodos HTTP para transferência de um estado representativo é o envio de dados como, temperatura, umidade e luminosidade, pelo sensor e o método PUT para o envio dos dados dos sensores. Outro exemplo é para obter os dados das medições, o método GET pode ser usado pelo servidor, como é mostrado na Figura 8.

GET /sensor/data HTTP/1.1

Figura 8 – Requisição http. LEMOS, 2014.

Desenvolver aplicações para IoT minimizando uso de memória e energia é de extrema necessidade, pois os objetos que compõem a rede possuem restrições de tamanho e bateria. Devido a esta necessidade o COAP (*Constrained Application Protocol*) foi projetado para reduzir o uso de largura de banda e complexidade de implementação. Segundo

Kovatsch, et al., (2011, p.856), é um protocolo de camada de aplicação que reduz o consumo de energia quando comparado ao uso do HTTP no projeto de uma *interface* REST. Ele adota padrões do HTTP como abstração de recurso, URI, interação RESTful e opções de cabeçalho extensíveis. E uma diferença importante é que o HTTP usa TCP como protocolo de transporte, mas o CoAP usa UDP (FIELDING, 2000, p. 55).

4.1 Definição do vocabulário

O vocabulário semântico utilizado para IoT, possui requisitos básicos necessários para extrair os resultados da comunicação com os diversos serviços ou aplicações. Os requisitos fundamentais que enumeram os conceitos a serem utilizados como bases para criação de um vocabulário padrão são:

- ✓ Clareza e objetividade: os termos devem ter definições objetivas e escritas em uma linguagem natural;
- ✓ Completeza: um termo deve expressar as condições necessárias para as necessidades de uma aplicação;
- ✓ Extensibilidade monôtonica: permite a inclusão de novos termos no vocabulário sem necessidade de revisão das definições, isto é, mantém sua generalidade;
- ✓ Mínimo compromisso no vocabulário: permite que sejam definidas tão poucas suposições quanto possíveis sobre o mundo a ser modelado;
- ✓ Distinção no vocabulário: as classes definidas no vocabulário devem ser disjuntas, declarando que duas coisas são diferentes, sem subrepor os seus conceitos.

Um dos maiores problemas observados na utilização de vocabulários é a necessidade do comprometimento em se criar vocabulários utilizando estruturas ontológicas já definidas. Como também reutilizá-las após serem publicadas. Mesmo que na criação desses vocabulários se tenha o cuidado em seguir alguns dos requisitos citados anteriormente, como clareza ou mínimo compromisso no vocabulário.

Logo, para a construção do vocabulário semântico devem-se seguir alguns passos:

- ✓ Identificar a finalidade do vocabulário: primeiro passo é definir o que se quer representar e de que forma se representará. Esse passo consiste do momento mais importante, onde se tem que ter em mente exatamente que parte do mundo real será representada. Esse processo requer muito cuidado e técnicas para que o domínio seja bem analisado e concebido;
- ✓ Construir o vocabulário: nesse ponto deve-se verificar a existência de vocabulários que possam auxiliar na construção, com a utilização de ferramentas de *software* e *hardware;*
- ✓ Análise: verificar se o vocabulário construído realmente desempenhará o papel esperado, e se retrata de forma fiel no mundo real;
- ✓ Documentação: tão importante quanto construir um vocabulário, é documentá-la de forma adequada, para que outras pessoas possam reutilizá-lo;
- ✓ Guia de referência: informando como utilizar o vocabulário em algumas aplicações da *web*.

O vocabulário construído, conforme a Figura 9 seguiu os padrões definidos anteriormente para que dessa maneira possa ter maior interação com aplicações na IoT e com os dispositivos. Um diagrama de classes foi utilizado para demonstrar a estrutura das classes de um sistema onde estas representam as "coisas" que são gerenciadas por uma aplicação desejada. Uma classe é a definição de um tipo de objeto e todo objeto é uma instância de classes. As classes podem se relacionar com outras através de diversas maneiras: associação (conectadas entre si), dependência (uma classe depende ou usa outra classe), especialização (uma classe é uma especialização de outra classe), ou em pacotes (classes agrupadas por características similares) (FOWLER, 2014, p.143).

Um sistema normalmente possui alguns diagramas de classes e certa classe pode participar de vários diagramas de classes. A implementação de uma classe num diagrama pode ocorrer utilizando-se uma linguagem de programação orientada a objetos que tenha suporte direto para construção de classes. A criação do diagrama de classes necessitou que as mesmas estivessem identificadas, descritas e relacionadas entre si.

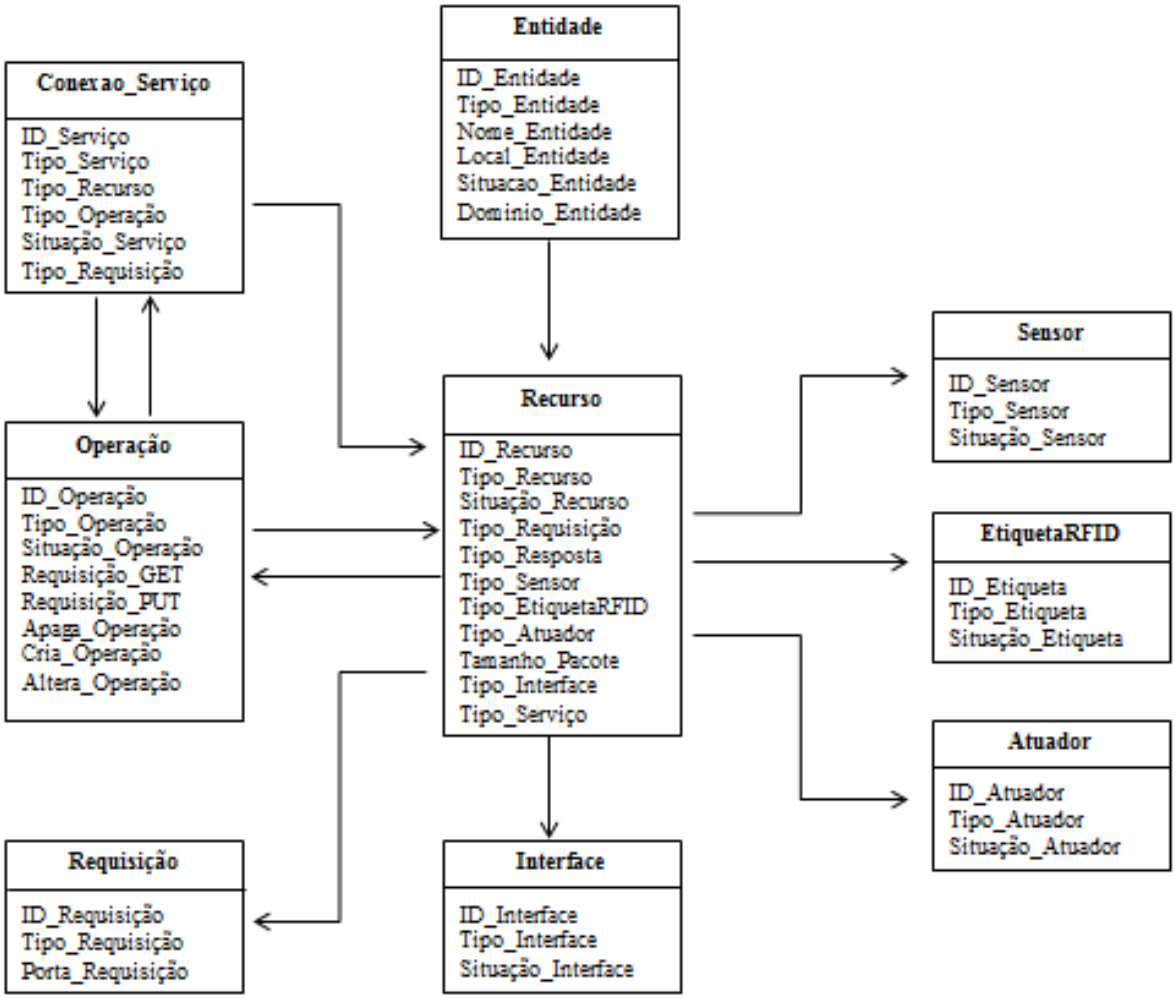

Figura 9 - Diagrama do vocabulário padrão. Elaborado pelo autor, 2016.

O diagrama do vocabulário padrão esta dividindo nas seguintes classes: Conexão_Serviço; Entidade; Recurso; Operação; Sensor; EtiquetaRFID; Requisição; Interface; e Atuador. Cada classe tem suas propriedades definidas, conforme a Tabela 3, necessárias para realização do início de uma conexão do serviço a ser descoberto na IoT. Cada conexão de serviço necessita de um recurso, que pode ser um sensor, uma etiquetaRFID ou atuador. O recurso poderá realizar uma requisição e/ou operação, além de poder solicitar uma interface a ser executado.

4.2 Descrição das Classes e Propriedades

As classes foram descritas na Tabela 2 e as propriedades de cada objeto das classes na Tabela 3.

Tabela 2 – Definição das Classes do Vocabulário.

Nome da Classe	Campo	Comentário	Refinamento	Propriedades Relacionadas
ServicoIoT:ConexaoServiço	Conexão do serviço	A conexão usada para a transferência de um Serviço da IoT.	ServicoIoT:Operação	ID_Serviço Tipo_Serviço Tipo_Recurso Tipo_Operação Situação_Serviço Tipo_Requisição
ServicoIoT:Entidade	Cabeçalho Entidade	Um cabeçalho de entidade em uma mensagem de Serviço da IoT	-	ID_Entidade Tipo_Entidade Nome_Entidade Local_Entidade Situacao_Entidade Dominio_Entidade
ServicoIoT:Recurso	Cabeçalho do Recurso	Um cabeçalho do recurso em uma mensagem de Serviço da IoT.	ServicoIoT:ConexaoServiço; ServicoIoT:Entidade; ServicoIoT:Operação	ID_Recurso Tipo_Recurso Situação_Recurso Tipo_Requisição Tipo_Resposta Tipo_Sensor Tipo_EtiquetaRFID Tipo_Atuador Tamanho_Pacote Tipo_Interface Tipo_Serviço
ServicoIoT:Operação	Cabeçalho da operação	Um cabeçalho da operação utilizada pelo serviço e/ou recurso.	ServicoIoT:ConexaoServiço; ServicoIoT:Recurso	ID_Operação Tipo_Operação Situação_Operação Requisição_GET Requisição_PUT Apaga_Operação Cria_Operação Altera_Operação
ServicoIoT:Sensor	Cabeçalho do Sensor	Um cabeçalho do sensor do utilizado pelo recurso	ServicoIoT:Recurso	ID_Sensor Tipo_Sensor Situação_Sensor
ServicoIoT:EtiquetaRFID	Cabeçalho da EtiquetaRFID	Um cabeçalho da etiqueta RFID utilizada pelo recurso	ServicoIoT:Recurso	ID_Etiqueta Tipo_Etiqueta Situação_Etiqueta
ServicoIoT:Atuador	Cabeçalho	Um cabeçalho do	ServicoIoT:Recurso	ID_Atuador

Nome da Classe	Campo	Comentário	Refinamento	Propriedades Relacionadas
	do atuador	atuador utilizado pelo recurso		Tipo_Atuador Situação_Atuador
ServicoIoT:Interface	Cabeçalho da Interface	Um cabeçalho da interface utilizada pelo recurso	ServicoIoT:Recurso	ID_Interface Tipo_Interface Situação_Interface
ServicoIoT:Requisição	Cabeçalho da Requisiçã o	Um cabeçalho da requisição utilizada pelo recurso	ServicoIoT:Recurso	ID_Requisição Tipo_Requisição Porta_Requisição

Fonte: Elaborado pelo autor, 2016.

Tabela 3 – Definição das Propriedades das Classes do Vocabulário.

Nome da Propriedade	Campo	Comentário	Domínio
ServicoIoT:ID_Serviço	Identificação do serviço	Identificação do serviço solicitado pela IoT.	ServicoIoT:Conexao_Servic o
ServicoIoT:Tipo_Serviço	Tipo do serviço	Tipo de serviço solicitado pela IoT	ServicoIoT:Conexao_Servic o
ServicoIoT:Tipo_Recurso	Tipo do Recurso	Tipo do recurso solicitado pelo serviço na IoT.	ServicoIoT:Recurso
ServicoIoT:Tipo_Operação	Tipo da operação	Tipo da operação utilizada pelo serviço e/ou recurso.	ServicoIoT:Operação
ServicoIoT:Situação_Serviço	Situação do serviço	Situação do serviço utilizado pelo recurso	ServicoIoT:Conexao_Serviç o
ServicoIoT:ID_Entidade	Identificação da entidade	Identificação da entidade que requer um recurso	ServicoIoT:Entidade
ServicoIoT:Tipo_Entidade	Tipo da entidade	Tipo da entidade que reque um recurso	ServicoIoT:Entidade
ServicoIoT:Local_Entidade	Localização da entidade	Localização da entidade que requer um recurso	ServicoIoT:Entidade
ServicoIoT:Situacao_Entidade	Situação da entidade	Situação da entidade que requer um recurso.	ServicoIoT:Entidade
ServicoIoT:Dominio_Entidade	Domínio da entidade	O domínio que a entidade está localizada.	ServicoIoT:Operação
ServicoIoT:ID_Operação	Tipo da operação	Tipo da operação a ser realizada pelo recurso e serviço.	ServicoIoT:Operação
ServicoIoT:Situação_Operação	Situação da operação	Situação da operação a ser realizada pelo recurso e serviço.	ServicoIoT:Operação
ServicoIoT:Requisição_GET	Requisição GET	Requisição GET solicitada pela pelo recurso e serviço.	ServicoIoT:Operação

Nome da Propriedade	Campo	Comentário	Domínio
ServicoIoT:Requisição_PUT	Requisição PUT	Requisição PUT solicitada pela pelo recurso e serviço.	ServicoIoT:Operação
ServicoIoT:Apaga_Operação	Apaga operação	Operação que apaga uma requisição ou resposta do recurso ou serviço.	ServicoIoT:Operação
ServicoIoT:Altera_Operação	Altera operação	Operação que altera uma requisição ou resposta do recurso ou serviço.	ServicoIoT:Operação
ServicoIoT:Cria_Operação	Cria operação	Operação que cria uma requisição ou resposta do recurso ou serviço.	ServicoIoT:Operação
ServicoIoT:ID_Recurso	Identificação do recurso	Identificação do recurso solicitado pela entidade.	ServicoIoT:Recurso
ServicoIoT:Situação_Recurso	Situação do recurso	Situação do recurso solicitado pela entidade.	ServicoIoT:Recurso
ServicoIoT:Tipo_Resposta	Tipo da resposta do recurso	Tipo da resposta do recurso solicitado pela entidade.	ServicoIoT:Recurso
ServicoIoT:Tipo_Sensor	Tipo do sensor	Tipo do sensor utilizado pelo recurso.	ServicoIoT:Sensor
ServicoIoT:Tamanho_pacote	Tamanho do pacote	Tamanho do pacote (informação) a ser solicitado pelo recuro.	ServicoIoT:Recurso
ServicoIoT:Tipo_Interface	Tipo de interface	Tipo de interface requerida pelo recurso	ServicoIoT:Interface
ServicoIoT:ID_Interface	Identificação da interface	Identificação da interface requerida pelo recurso.	ServicoIoT:Interface
ServicoIoT:Situação_Interface	Situação da interface	Situação da interface requerida pelo recurso.	ServicoIoT:Interface
ServicoIoT:ID_Requisição	Tipo de requisição	Tipo da requisição solicitada pelo recurso	ServicoIoT:Requisição
ServicoIoT:Porta_Requisição	Porta da Requisição	Porta da requisição solicitada pelo recurso	ServicoIoT:Requisição
ServicoIoT:ID_Sensor	Identificação do sensor	Identificação do sensor utilizado pelo recurso	ServicoIoT:Sensor
ServicoIoT:Situação_Sensor	Situação do sensor	Situação do sensor utilizado pelo recurso	ServicoIoT:Sensor
ServicoIoT:ID_Etiqueta	Identificação da etiqueta	Identificação da etiqueta utilizada pelo recurso.	ServicoIoT:EtiquetaRFID
ServicoIoT:Tipo_Etiqueta	Tipo da etiqueta	Tipo da etiqueta utilizado pelo recurso	ServicoIoT:EtiquetaRFID
ServicoIoT:Situação_Etiqueta	Situação da etiqueta	Situação da etiqueta utilizado pelo recurso.	ServicoIoT:EtiquetaRFID

Nome da Propriedade	Campo	Comentário	Domínio
ServicoIoT:ID_Atuador	Identificação do atuador	Identificação do atuador utilizada pelo recurso.	ServicoIoT:Atuador
ServicoIoT:Tipo_Atuador	Tipo do atuador	Tipo do atuador utilizado pelo recurso	ServicoIoT:Atuador
ServicoIoT:Situação_Atuador	Situação do atuador	Situação do atuador utilizado pelo recurso	ServicoIoT:Atuador

Fonte: Elaborado pelo autor, 2016.

4.3 Descobrimento de serviço

O processo de descobrimento de serviço informa quais serviços estão disponíveis, solicitados por um dispositivo, para que outros dispositivos da rede possam usar estes serviços e até oferecer novos serviços, sem que a oferta seja pública. Então um mecanismo de descobrimento de serviço com REST *Web Services* foi ajustado para o vocabulário criado de acordo com as necessidades e limitações do ambiente.

O protocolo SLP (*Service Location Protocol*) é um protocolo de descoberta serviço que permite que computadores e outros dispositivos possa encontrar serviços em uma rede local sem configuração prévia. Foi projetado para escalar em pequenas redes, não gerenciados para redes de grandes empresas. Apesar da simplificação do SLP é mais compreensível e menos complexo que outros protocolos como UPnP (VITERBO, 2015, p.11) e Zeroconf (LEE, 2007, p.3) que são mais que protocolos de descobrimento de serviço.

O SLP é um protocolo de anúncio de serviço que possibilita a descoberta de serviços e anúncio por dispositivos na rede. Leve e simples quando comparado a protocolos como UPnP e Zeroconf, pode ser explicado brevemente pelos três componentes que o formam.

- ✓ UA – *User Agent* (Usuário Agente)
- ✓ SA – *Service Agent* (Serviço Agente)
- ✓ DA – *Directory Agent* (Diretório Agente)

O *User Agent* é o usuário agente que realiza requisições e não provê serviços. É quem solicita uma requisição ao serviço agente. Quando um UA é executado, os serviços do seu interesse são procurados na rede, um UA pode estar interessado em apenas um serviço ou ter um serviço principal. *Service Agent* é a classificação dada ao dispositivo que provê

serviços e responde requisições de agentes do usuário. Quando um SA é executado, os serviços disponibilizados são anunciados na rede, UAs interessados respondem ao anúncio e uma relação é estabelecida. *Directory Agent* é a classificação dada ao dispositivo que é autorizado por um SA para responder a requisições de um UA, DAs são importantes para a escalabilidade. Um SA pode ser sobrecarregado com muitas requisições, cabe ao DA responder requisições que iriam para o AS, conforme a Figura 10.

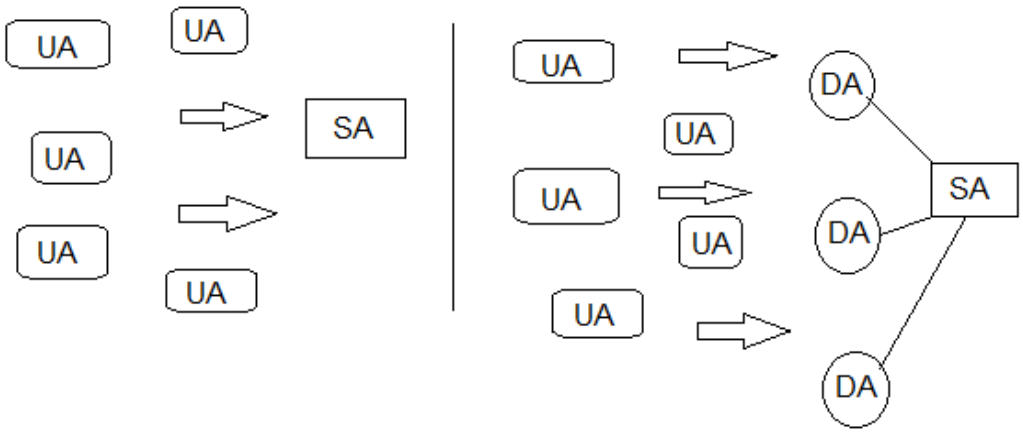

Figura 10 – DAS evitando sobrecarga do AS. LEMOS, 2014.

4.4 Ferramentas utilizadas

4.4.1 Linguagem de Programação JSON (*JavaScript Object Notation*)

JSON (*JavaScript Object Notation*) é um formato de dados leve, fácil de compreensão para ler e escrever, e para as máquinas analisar e gerar. É baseado em um subconjunto da linguagem de programação *JavaScript*, *Standard* ECMA-262 (ECMA, 2016). JSON é um formato de texto que é completamente independente do idioma, mas usa convenções que são familiares aos programadores da família da linguagem C, incluindo C, C ++, C #, *Java*, *JavaScript*, *Perl*, *Python*, e muitos outros. Estas propriedades fazem JSON uma linguagem de intercâmbio de dados ideal.

Conforme a RFC 4627, JSON pode representar quatro primitivos tipos (*strings*, números, booleanos e nulos) e dois tipos estruturados (objetos e matrizes) (CROCKFORD,

2006, p.8). A cadeia é uma sequência de zero ou mais caracteres *Unicode*. Esta especificação define JSON-LD, um formato baseado em JSON. Sua sintaxe é projetada para integrar facilmente em sistemas implantados que já usam JSON, e fornece um caminho de atualização suave do JSON para JSON-LD. Destinado principalmente para utilizar dados em ambientes de programação com base na *web*, construir serviços *web* interoperáveis e armazenar dados ligados em mecanismos de armazenamento baseados em JSON (WALMSLEY, 2016, p.29).

Provavelmente devido a essas propriedades e à concisão do formato, várias APIs *web* estão sendo construídas baseadas no formato JSON, que garante uma transição suave e simples de sistemas baseados em JSON, permitindo que as organizações que já implantaram grande infraestrutura baseada em JSON possam utilizar os recursos do JSON-LD de uma forma que não seja prejudicial para as suas operações do dia-a-dia e seja transparente para seus clientes atuais. Um exemplo básico de esquema neste formato pode ser observado na Figura 11.

```
{
 "@context":
 {
  "name": "http://schema.org/name",
  "image": {
   "@id": "http://schema.org/image",
   "@type": "@id"
  },
  "homepage": {
   "@id": "http://schema.org/url",
   "@type": "@id"
  }
 },
 "name": "Manu Sporny",
 "homepage": "http://manu.sporny.org/",
 "image": "http://manu.sporny.org/images/manu.png"
}
```

Figura 11 – Exemplo De Esquema Em Json

Figura 11 – Exemplo de esquema em JSON. SPORNY, 2014.

A simplicidade de interpretação e de criação do JSON facilita a criação de muitas APIs. O JSON.*simple* é um API necessária que pode ser baixada no seguinte endereço *web*: http://code.google.com/p/json-simple/. Essa API é bem pequena e possuem muitas funcionalidades, compatíveis com o padrão RFC 4627. Ao instalar esta biblioteca, poderá ser montada a classe que irá criar esse o arquivo JSON (SPORNY, 2014, p.16).

As classes que serão importadas são:

- *FileWriter* – Classe utilizada para se escrever em arquivos;
- *IOExceptio*n – Classe que lança uma exceção de entradas e saídas;
- *JSONObject* – Classe que cria objetos JSON (importada da biblioteca *JSON.simple*).

Cada uma dessas classes possuem outras subclasses ou dependências que poderão ser utilizadas de acordo com o que a aplicação necessitará. Ferramentas como o RDF, onde foi falado anteriormente neste capítulo, podem ser utilizadas com este padrão JSON.

4.4.2 Sistemas Operacional *Contiki*

O *Contiki* (SEHGAL, 2013, p.5) é um sistema operacional de código aberto que opera para nós de sensores, desenvolvido pelo Instituto de Computação da Suíça, para IoT. Algumas das suas principais características estão: carregamento e descarregamento dinâmico de código em tempo de execução; a possibilidade de uma programação concorrente em cima de um kernel orientado a eventos; utilizar os padrões de *Internet* como o IPv6 e o IPv4 com eficiência energética; e o suporte aos padrões de comunicação sem fio de baixo consumo de energia, como, CoAP *(Constrained Application Protocol), 6LowPAN (IPv6 Over Low Power Wireless Personal Area Networks) e RPL (Routing Protocol Low Power)*.

Reusing (2012, p.17) afirma que uma aplicação *Contiki*[1] consiste no *kernel*, nas bibliotecas, e no carregador do programa ou processos, e que estes processos são serviços ou programas desta aplicação. Porém possuem uma diferença onde a funcionalidade dos serviços pode ser usada em mais de um processo e as aplicações são usadas somente para outro processo, não oferecendo funcionalidades para processos distintos.

Com uma das características de possuir a capacidade de substituir todos os programas de forma dinâmica em tempo de execução, realizar a função de realocação dinâmica, a qual muda o programa com a ajuda das informações de realocação que estão presentes no sistema binário. Após o carregamento do programa, executa a função de inicialização, onde um ou mais processos podem ser iniciados.

O núcleo do sistema operacional *Contiki* é formado pelo *kernel*, bibliotecas, *driver* e o carregador de programa, como podem ser vistos na Figura 12. O núcleo é geralmente implementado como um único binário, enquanto que os programas podem ser carregados de forma independente (REUSING, 2012, p.19).

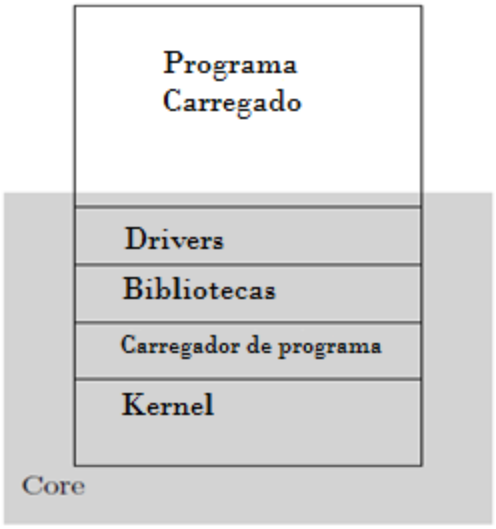

Figura 12 – Estrutura do *Contiki*. REUSING, 2012.

Para que possa acessar o *hardware* deve ser implementadas bibliotecas e *drivers* para realizar a comunicação, pois o *kernel* desse sistema operacional não oferece nenhuma abstração de *hardware*. E todos os componentes de um aplicativo *Contiki* têm acesso direto ao *hardware* subjacente.

Para realizar um controle de energia devem ser criadas aplicações que possam gerenciar o tamanho da fila de eventos que o *Contiki* fornece, e caso esteja vazia, dar a possibilidade aos dispositivos de colocar o microcontrolador ou os periféricos de *hardware* em modo ocioso. Com isso, o *Contiki* por sua vez é uma melhor escolha quando se precisa

de mais flexibilidade, por exemplo, se um nó da rede precisar ser frequentemente atualizado (REUSING, 2012, p.22).

4.4.3 Simulador *Cooja*

O *Cooja Network Simulator* é um simulador de rede *Contiki*, faz com que o desenvolvimento e depuração de *software* para redes sem fio grandes seja mais fácil. Fornece um ambiente de simulação que permite aos desenvolvedores tanto ver seus aplicativos como são executados em redes de larga escala ou em dispositivos de *hardware* totalmente emulados.

O código pode ser encontrado na pasta *Contiki*/tools/*Cooja*/, que está localizado no sistemas operacional *Contiki* 2.6. Ele utiliza uma combinação de código Java para a *interface* de *front-end* e plataforma específica de emuladores para realizar as simulações (SEHGAL, 2013, p.6).

Na Figura 13 pode ser observada a interface gráfica do simulador *Cooja*, onde pode compilar, construí e iniciar uma simulação.

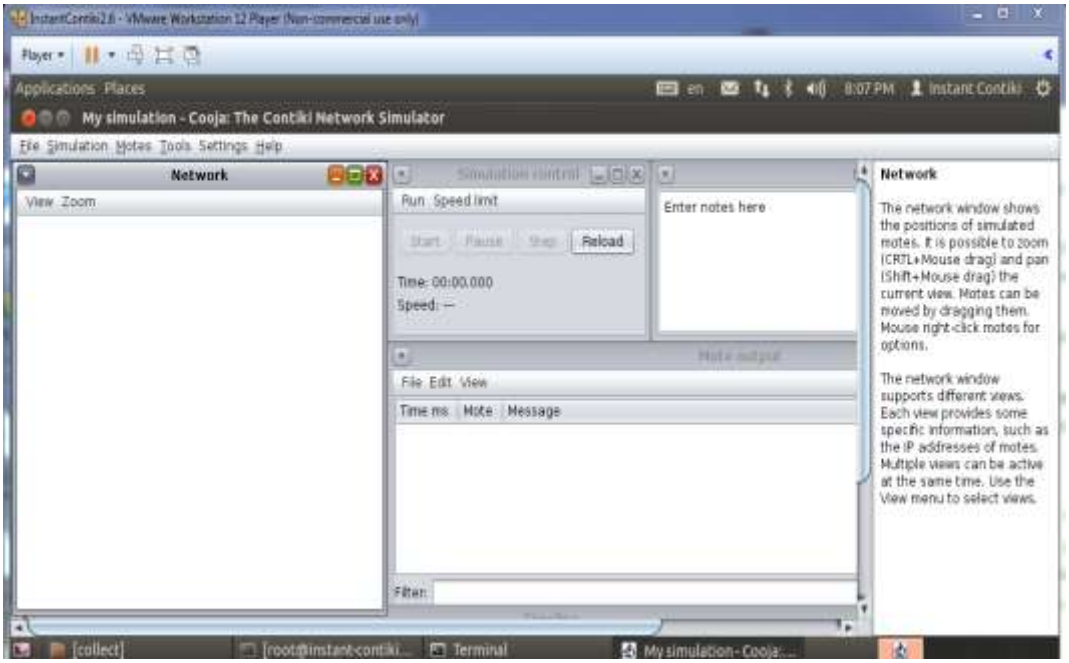

Figura 13 – Tela inicial do simulador *Cooja*. Elaborado pelo autor, 2016.

Para a implementação do vocabulário foi utilizada a linguagem JSON e na sua validação o simulador *Cooja* contido no sistema operacional *Contiki*, sistema de código aberto que conecta micro controladores de baixa potência à *internet*. O *Contiki* utiliza os padrões IPv6 e IPv4 e os últimos padrões *wireless* de baixo consumo: 6LoWPAN , RPL, COAP. É executado em uma grande quantidade de dispositivos sem fio de baixa potência e as aplicações podem ser desenvolvidas em C, Java, JSON, por exemplo, através do *Contiki*.

Algumas das características importantes do *Contiki* são: projetado para trabalhar ocupando pouco espaço de memória; rede IP completa, com protocolos como o UDP, TCP, HTTP, 6LoWPAN, RPL, COAP e IPv6; opera em sistemas de baixo consumo de energia; o sistema de compilação *Contiki* torna mais fácil compilar aplicativos para qualquer uma das plataformas *Contiki* disponíveis; e com isso facilita o teste de aplicativos em uma variedade de plataformas (SEHGAL, 2013, p.4).

O simulador *Cooja* é baseado em *Java*, que foi projetado para redes de sensores sem fio e é executado no sistema operacional *Contiki*. Simula redes de nós sensores, onde cada nó pode ser de um tipo diferente, diferindo tanto no *software* quanto no *hardware*. Muitas partes do simulador podem ser facilmente substituídas ou estendidas com novas funcionalidades (ÖSTERLIND, et al., 2006). Um nó simulado em *Cooja* tem três propriedades básicas: sua memória de dados, o tipo de nó e seus periféricos de *hardware*.

O tipo de nó pode ser compartilhado entre vários nós e determina a propriedades comuns de todos esses nós. Todas as interações com simulações e nós simuladas são realizadas através de *plug-ins*. Um exemplo de *plug-in* é um onde possibilita um usuário iniciar ou parar a simulação. *Interfaces* e *plug-ins* podem ser facilmente adicionados ao simulador, com isso facilitando aos usuários a personalização da sua simulação (ÖSTERLIND, et al., 2006). Qualquer nó simulado em *Cooja* pertence a um tipo de nó. O tipo de nó determina, entre outros, que tipo de aplicativo *Contiki* irá simular. Para criar um nó, clica-se no *Menu Motes -> Add Motes -> Create New Mote Type -> Sky Mote*. A Figura 14 mostra como se deve adicionar um Sky *mote* a uma nova simulação no *Cooja*. Um *Sky mote* é um sensor sem fio de baixo consumo de energia que utiliza padrões da indústria como *usb* (*Universal Serial Bus*) e o padrão para redes *wireless* IEEE 802.15.4 para interoperar com outros dispositivos, além de servir para a sua programação (SEHGAL, 2013, p.4).

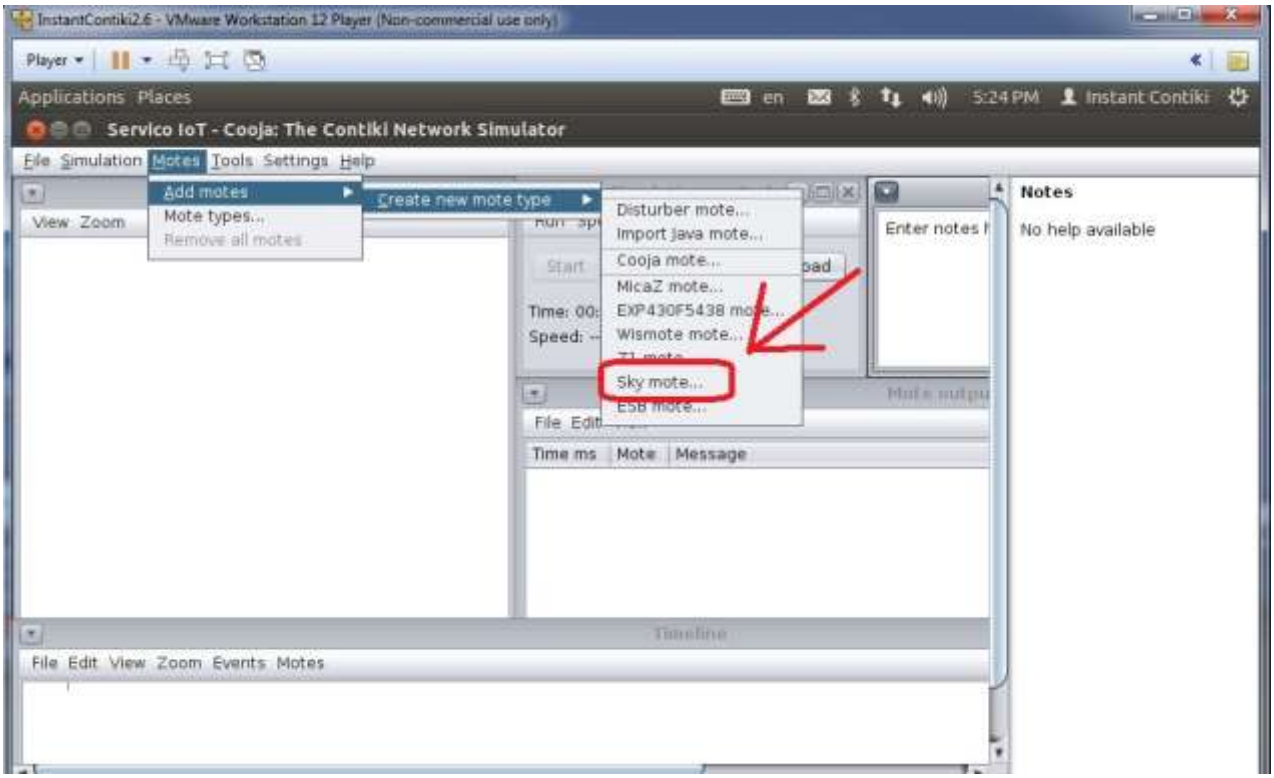

Figura 14 – Tela de criação de um nó do tipo Sky Mote. Elaborado pelo autor, 2016.

5 IMPLEMENTAÇÃO DO VOCABULÁRIO SEMÂNTICO

Até o presente momento foram vistas as técnicas utilizadas para a construção do vocabulário e como implementá-lo. Neste capítulo será apresentada a implementação do vocabulário semântico.

Utilizou-se como base um mecanismo para descobrimento dos serviços na IoT, sendo modificado para a estrutura do vocabulário proposto e o simulador *Cooja* para os testes a serem realizados na arquitetura do serviço, no qual foram simulados três *TmoteSky* que possuem 10KB de RAM e 48KB de ROM.

5.1 Implementação do Vocabulário

A implementação do vocabulário padrão para descobrimento dos serviços na IoT foi embasado no mecanismo de descoberta feito por Lemos (2014). O mecanismo foi adaptado para que o vocabulário criado pudesse realizar um comportamento básico baseado nas unidades básicas do SLP: UA, SA e DA. A aplicação Melhor Preço, simula um objeto instalado num refrigerador ou qualquer outro dispositivo que possa armazenar produtos. O objeto efetua compras no servidor Melhor Preço sem necessitar pesquisar preços dos produtos desejados na busca. Esta primeira ação é o descobrimento do serviço para achar os Mercados que possuem o produto desejado. Ao chegar uma requisição de compra o servidor Melhor Preço procura o melhor preço do produto entre os diversos mercados cadastrados no servidor garantindo o melhor preço para o objeto no dispositivo.

Os Mercados utilizam o procedimento de descobrimento de serviço para que os melhores preços sejam cadastrados no servidor de Melhor Preço. Esta comunicação pode ser vista na Figura 15 que demonstra um exemplo da utilidade de descobrimento de serviço, onde o servidor Melhor Preço oferece o serviço de pesquisa e cadastra os mercados para que seja realizada a compra com o melhor preço do produto desejado pelo objeto.

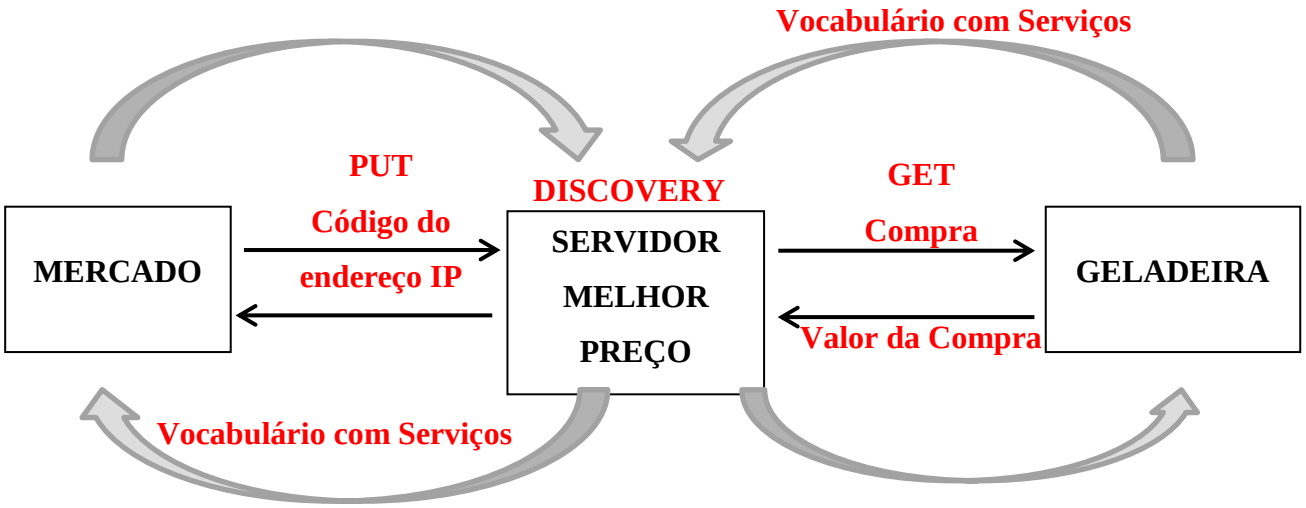

Figura 15 – Comunicação da aplicação melhor preço. Elaborado pelo autor, 2016.

O vocabulário padrão foi utilizado como modelo para a elaboração de um vocabulário para descoberta de serviços em um servidor da aplicação de menor preço, conforme pode ser visto o código no Apêndice A. À aplicação Melhor Preço é composta por dois clientes e um servidor. Os clientes são uma geladeira e um supermercado e o servidor é o responsável por gerir os menores preços de produtos oferecidos por supermercados. O servidor foi elaborado para comportar dois serviços: o cadastro de supermercados e a consulta de preços. Os serviços foram simulados no *Cooja* com um nó sensor representando um supermercado, um nó sensor representado uma geladeira, e um nó sensor representando um servidor de menor preço de acordo com a Figura 16. Todo código-fonte das aplicações foram disponibilizados em um repositório chamado de *GitHub* (LIMA, 2016).

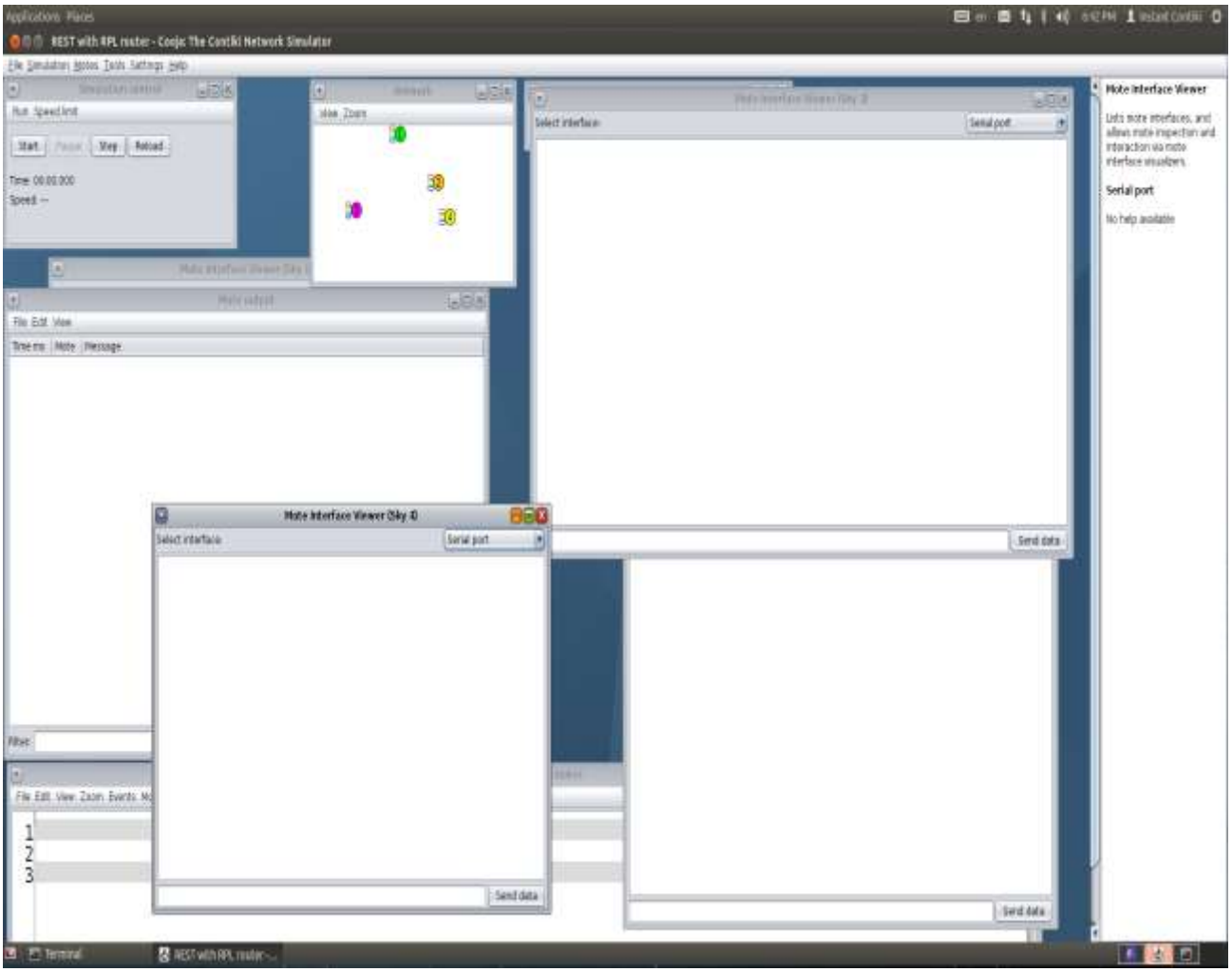

Figura 16 - Início da comunicação dos nós simulados. Elaborado pelo autor, 2016.

O cadastro de supermercados é um serviço *Representational State Transfer* (REST) do tipo PUT. Nele, o servidor de Melhor Preço recebe o código de um supermercado, o endereço IPv6 de um supermercado e uma lista contendo códigos, quantidades e preços de produtos e o armazena em um vetor. O servidor retorna 1 caso tenha armazenado corretamente as informações, ou 0 em caso contrário de acordo com as linhas 9 e 12 do trecho do código descrito na Figura 17.

```
1   // So insere um mercado se ele ainda nao existir
2   if (found == 0)
3   {
4   // Insere um mercado no vetor
5   uip_ip6addr (&mercados [quantidadeMercados].ip, atoi (ip_a), atoi (ip_b), atoi (ip_c), atoi (ip_d), atoi
    (ip_e), atoi
6   (ip_f), atoi (ip_g), atoi (ip_h));
7   mercados [quantidadeMercados++].cod = codigo;
8   printf("[MELHOR PRECO] (discovery_mercado) Inseriu o mercado de codigo %d\n", codigo);
9   coap_set_payload (response, "1", 1); // Envia a resposta 1 para informar que a inclusao do mercado ocorreu
10   }
11  else{
12  coap_set_payload (response, "0", 1); // Envia a resposta 0 para informar que a inclusao do mercado nao
13  ocorreu
```

Figura 17 - Trecho de código do servidor mercado. Elaborado pelo autor, 2016.

A consulta de preços é um serviço REST do tipo GET. A geladeira consulta o servidor de Melhor Preço através deste serviço, enviando o código e a quantidade do produto para serem consultados. O servidor de Melhor Preço pesquisa no vetor estas informações e retorna à geladeira o código do supermercado que contém a quantidade requerida e o menor preço do produto consultado.

Antes de utilizar os serviços, uma geladeira ou um supermercado precisa fazer uma requisição GET para recuperar o arquivo em *JavaScript Object Notation* (JSON) contendo os serviços disponíveis para saber suas descrições, pontos de chamada, parâmetros com os respectivos formatos e mensagens de retorno. A estrutura em JSON dos serviços foi criada de acordo com o Apêndice B.

5.2 Consumo Energético e de Memória

O teste realizado para o vocabulário semântico definido ficou incompleto devido aos nós sensores somente conseguirem enviar no máximo 64 *bytes* de dados. Isso faz com que as

mensagens enviadas utilizando a *Application Programming Interface* (API) *Erbium* ou *Constrained Application Protocol* (CoAP), sejam truncadas antes de serem enviadas na camada de enlace para evitar erros que possam acontecer na camada física. Isso é evidenciado na exibição dos conteúdos das mensagens da aplicação de Melhor Preço quando, por exemplo, um supermercado ou uma geladeira envia a requisição de descoberta de serviços ao servidor de Melhor Preço para receber o arquivo em JSON contendo a especificação dos serviços lá existentes.

A requisição de descoberta de serviços, enviada a um servidor de Melhor Preço, advinda de um cliente geladeira ou supermercado é uma mensagem *Hyper-Text Transfer Protocol* (HTTP) REST do tipo GET. A mensagem de requisição é endereçada ao *Universal Resource Locator* (URL) denominado *discover* do servidor e não necessita de parâmetros.

A mensagem de requisição de descoberta de serviços é recebida no servidor de Melhor Preço que a responde com um arquivo JSON no formato do vocabulário padrão definido nesta dissertação. Este arquivo contém a especificação dos serviços constantes no servidor de Melhor Preço. Esta mensagem de resposta contém 1305 *bytes* de dados e é enviada de volta ao cliente que solicitou a descoberta de serviços. A resposta é truncada para 64 *bytes* antes de ser enviada. Essa é uma restrição física dos nós de uma rede simulada no *Contiki*, Figura 18. Desta forma, o cliente não recebe o arquivo com as especificações dos serviços contidos no servidor de Melhor Preço.

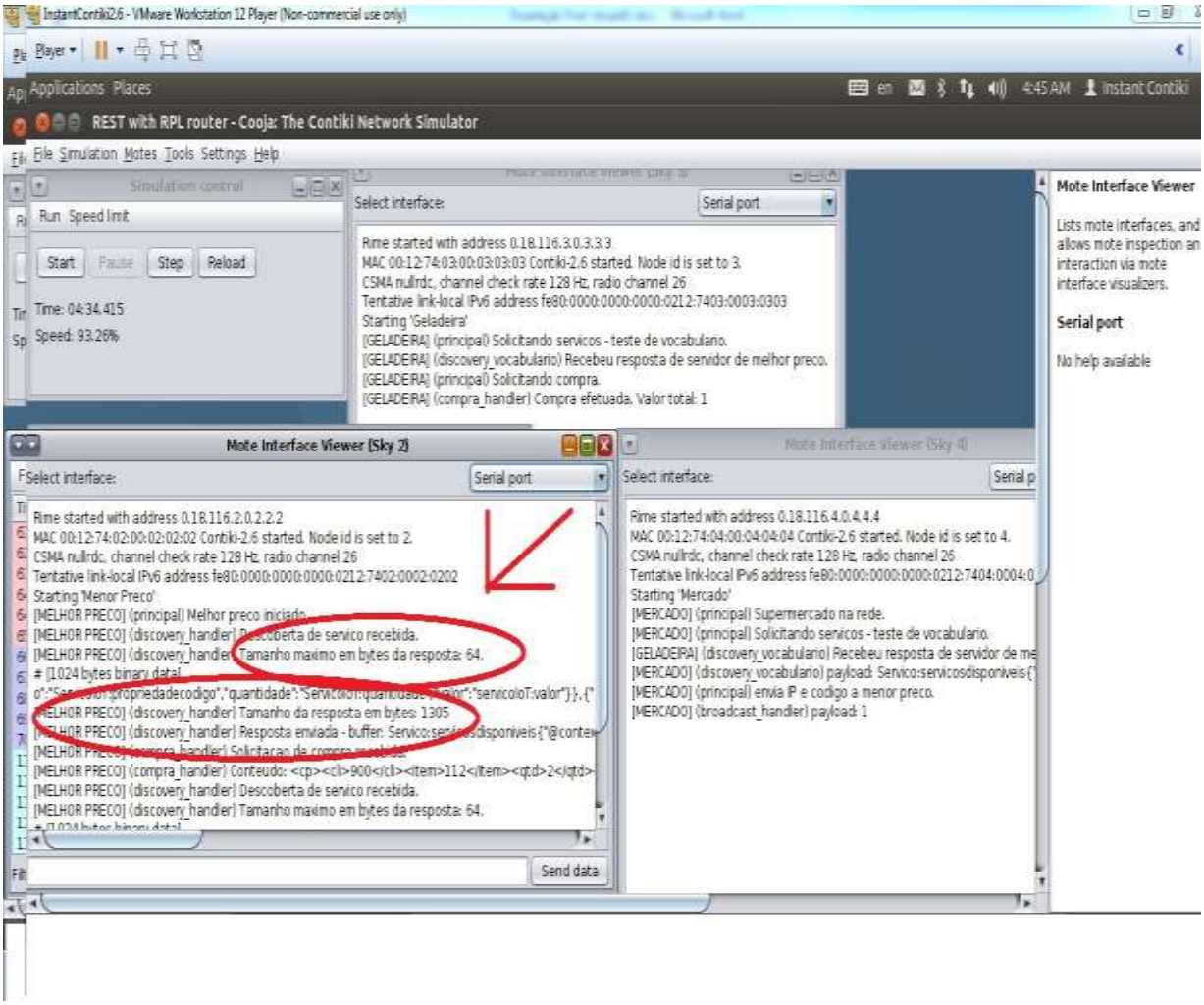

Figura 18 - Tamanho da mensagem de resposta. Elaborado pelo autor, 2016.

Duas alternativas foram realizadas para fazer com que os nós sensores do *Contiki* enviassem mais que 64 *bytes* de dados, mas ambas não executaram e não mostrariam a realidade de uma comunicação real de um nó sensor simulado pelo *Contiki*. As alternativas foram:

I. Aumentar o tamanho dos dados enviados e recebidos na camada de aplicação pela API REST;
II. Aumentar o tamanho do *buffer* de envio e recepção de quadros nos nós sensores simulados.

A primeira alternativa, aumentar o tamanho dos dados enviados e recebidos pela API REST, não funcionou porque a limitação ocorre na camada de enlace dos nós sensores.

Portanto, a segunda alternativa foi tentada combinada com a primeira. Porém aumentar o tamanho do *buffer* de envio e recepção dos nós sensores implicou em utilizar mais memória, outro recurso limitado, e, quando pareceu funcionar, gerou efeitos colaterais na troca de dados da aplicação, fazendo-a funcionar de forma errática na troca de mensagens entre clientes e servidor. Toda esta comunicação entre os nós sensores pode ser observada na Figura 19.

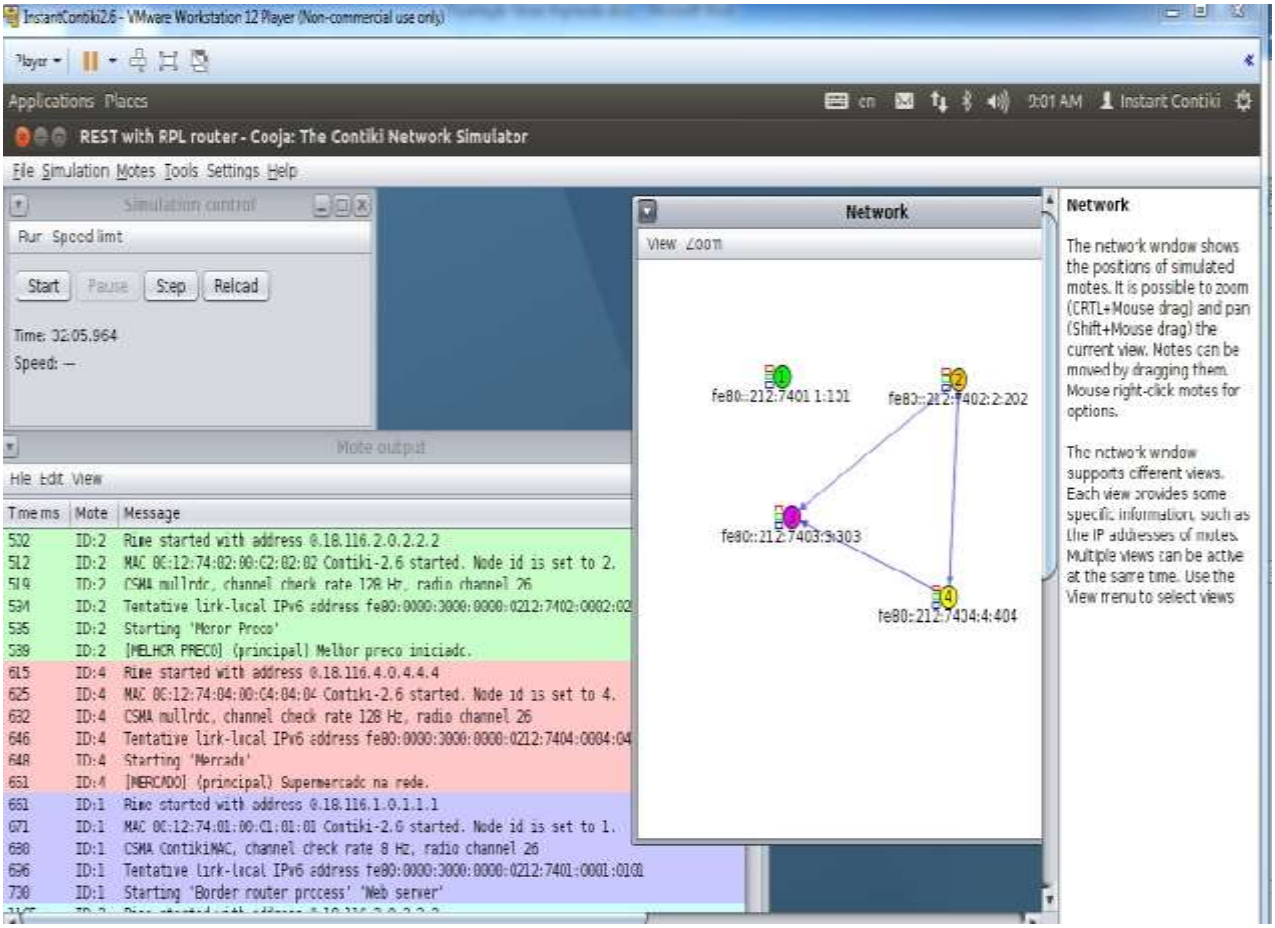

Figura 19 - Comunicação entre os nós sensores. Elaborado pelo autor, 2016.

6 CONCLUSÕES

O desenvolvimento do presente trabalho possibilitou a construção de um vocabulário semântico para descobrir serviços na IoT e realizar sua implementação no *Contiki* com o simulador *Cooja* com três nós sensores *TmoteSky*, onde trocam mensagens através de dois serviços PUT e GET enviando o vocabulário nas respostas.

O vocabulário pode ser visto como um ponto determinante em questões relativas à área de integração de informação. Sua capacidade em reconhecer conceitos e relacioná-los entre si torna-o poderoso na resolução de problemas relativos à heterogeneidade semântica. Nos trabalhos correlacionados foram mostradas algumas construções de vocabulários para aplicações específicas, relacionando suas caraterísticas e comparando-as, como por exemplo, a utilização de semântica e serviços.

Existem diferentes formas de empregar os vocabulários, sejam para descobrimento de um determinando serviço ou para selecionar vários serviços dentro de uma rede *web* sem distinção de vocabulários e seus termos para a correta comparação. Levando em consideração essa necessidade de um vocabulário semântico construído para a descoberta de serviços na IoT, foi realizada a sua implementação utilizando como base um mecanismo de busca já existente, mas adaptando-o para que seu comportamento funcionasse na busca de serviços para a IoT, e não somente em endereços IP’s.

Para a implementação criou-se uma aplicação Melhor Preço, onde foi criado um servidor de Melhor Preço, um Mercado e uma Geladeira como objeto de buscas de serviços utilizando o vocabulário construído. Primeiramente foi utilizada uma parte do vocabulário no simulador *Cooja* para que pudesse se mensurado o consumo de energia e memória na busca dos serviços. Mas, para esta análise não obtivemos um resultado satisfatório devido a limitação dos nós sensores do simulador, o *TmoteSky*, que possui de memória ROM somente 48 *bytes* e o vocabulário necessitava mas que 64 *bytes* para enviar a resposta dos dados.

Mesmo realizando as duas alternativas para que enviassem mais de 64 *bytes* que foram: aumentar o tamanho dos dados enviados e recebidos pela API REST; e aumentar o tamanho do *buffer* de envio e recepção nos nós sensores simulados, não foi obitida uma comunicação real de um nó sensor simulado pelo *Contiki*. Diante disso, as especificações do tamanho das mensagens somente foi descoberta ao ser realizada a simulação do vocabulário,

que é enviado apenas uma vez por cliente na descoberta dos serviços, não tendo envios repetidos.

Para trabalhos futuros, sugere-se que essa implementação seja levada para o mundo real utilizando dispositivos da IoT e realizando testes locais para a análise precisa dos serviços buscados, validando o vocabulário como um todo em uma estrutura física e não simulada, para que possamos obter resultados mais concretos e claros. Outro trabalho futuro seria a construção de uma API própria para o envio da resposta dos dados com o vocabulário padrão definido para descobrir serviços na IoT. Uma API que possa enviar o vocabulário semântico de descoberta de serviços em nós sem restrição de memória e envio de dados. Para isso, seria necessário um protocolo para seja feita a troca de mensagens não tendo limitações de memória entre os nós sensores.

REFERÊNCIAS

AGGARWAL, C.; ASHISH, N.; et al. T**he *Internet of Things*: A Survey from The Data-Centric Perspective**. In: Managing and Mining Sensor Data. Springer, Estados Unidos, cap. 12, p.383-428, 2013.

ATZORI, L.; IERA, A.; MORABITO, G. **The *Internet of Things*: A survey.** Italia. Computer Networks, v.54, p. 2787–2805, 2010.

BARNAGHI, P.; PRESSER, M. **Publishing linked sensor data**. In: Proceedings of the 3rd International Conference on *Semantic* Sensor Networks, CEUR-WS. org, v.668, p.1-16, 2010.

BAUER, M.; MEISSNER, S.; et al. **Service modelling for the *Internet of Things***. In: Computer Science and Information Systems (FedCSIS). Federated Conference on IEEE, p.949-955, 2011.

BITTENCOURT, I.; COSTA, E. **Modelos e ferramentas para a construção de sistemas educacionais adaptativos e semânticos.** Revista Brasileira de Informática na Educação, v.19, p.85, 2011.

CROCKFORD, D. **The application/json media type for javascript object notation (JSON).** RFC 4627. 2006.

ECMA-262 2016. **ECMA Script 2016 Language Specification**. 7a edição. ECMA International. Disponível em: <http://www.ecma-international.org/publications/ standards/Ecma- 262.htm> . Acesso em: 15 de jul. 2016.

FERRARIO, R.; GUARINO, N. **Towards an ontological foundation for services science**. In Future *Internet* Symposium, Springer Berlin Heidelberg, p.152-169, 2008.

FENSEL, D.; FACCA, F. M.; SIMPERL, E.; TOMA, I. **Semantic *Web***. In *Semantic Web Services*, Springer Berlin Heidelberg, p.87-104, 2011.

FIELDING, R. T. **Architectural styles and the design of network-based *software* architectures**. 2000. 180 f. Tese (Doutorado em filosofia) - University of California, Irvine. 2000.

FOWLER, M. **UML Essencial:** um breve guia para linguagem padrão. Bookman Editora, v. 3, p.1-162, 2014.

GIGLI, M.; KOO, S. ***Internet of Things*:** services and applications categorization. Advances in *Internet of Things*, v. 1, n. 02, p. 27, 2011.

GUTTMAN, E. **Service location protocol:** Automatic discovery of IP network services. IEEE *Internet* Computing, v. 3, n. 4, p.71-80, 1999.

GRUBER, T. R. **What is an ontology?**. Springer-Verlag, Knowledge Acquisition, v. 5, p.199-220, 2009.

HACHEM, S.; TEIXEIRA, T.; ISSARNY, V. **Ontologies for the *Internet of Things*.** INRIA Paris-Rocquencourt. HAL Id: hal-00642193, p.3, 2011.

JONES, S. **Toward an acceptable definition of service [service-oriented architecture]**. IEEE *software*, v.22, p.87-93, 2005.

KALAOJA, J.; KANTOROVITCH, J.; CARRO, S.; MIRANDA, J. M.; RAMOS, Á.; PARRA, J. **The Vocabulary Ontology Engineering for the *Semantic* Modelling of Home Services**. In ICEIS, v.3, p.461-466, 2006.

KILJANDER, J.; D'ELIA, A.; MORANDI, F.; HYTTINEN, P.; TAKALO-MATTILA, J.; YLISAUKKO-OJA, A.; CINOTTI, T. S. ***Semantic* interoperability architecture for pervasive computing and *Internet of Things***. IEEE access, v.2, p.856-873, 2014.

KNUBLAUCH, H.; FERGERSO, R. W.; NOY, N. F.; MUSEN, M. A. **The Protégé OWL plugin:** An open development environment for *semantic web* applications. In International *Semantic Web* Conference, Springer Berlin Heidelberg, p.229-243, 2004.

KOVATSCH, M.; DUQUENNOY, S.; DUNKELS, A. **A low-power CoAP for *Contiki***. In: 2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems. IEEE, p. 855-860, 2011.

LANTHALER, M.; GÜTL, C. **Hydra: A Vocabulary for Hypermedia-Driven *Web* APIs**. LDOW, v. 996, p.1-6, 2013.

LEE, J. W.; et al. **Z2Z**: Discovering zeroconf services beyond local link. IEEE Globecom Workshops. IEEE, p.1-7, 2007.

LEMOS, A. R. **Mecanismo dinâmico de descobrimento de serviços de alto nível para *Internet* das Coisas usando REST *Web* Services**. Programa Institucional de Bolsas de Iniciação Científica – PIBIC. UFS . 2014.

LIEBERMAN, J.; SINGH, R.; GOAD, C. **W3c geospatial vocabulary**. Incubator group report, W3C, p.1-13, 2007.

LIMA, M. de S. **Discovery Services**. Disponível em: https://github.com/MaykaLima/DiscoveryServices. Acesso em 20 de julho de 2016.

MACKENZIE, C. M.; LASKEY, K.; MCCABE, F.; BROWN, P. F.; METZ, R.; HAMILTON, B. A. **Reference model for service oriented architecture 1.0.** OASIS standard, v.12, p.1-31, 2006.

MASUOKA, R.; et al. **Ontology – Enabled Pervasive Computing Applications**. University of Maryland. 2003 IEEE, v.18, n.5, p.68-72, 2003.

MAYER, S.; GUINARD, D.; TRIFA, V. **Searching in a *Web*-based infrastructure for smart things. In *Internet of Things (IOT)***, *2012 3rd International Conference on the*, IEEE, p. 119-126, 2012.

MOURA, A. **A *Web* Semântica: Fundamentos e Tecnologias**. IEEE Computer Society, p.68-72, 2003.

NCE Tourism – F. N. ***Semantic markup* report Microformats, RDFA, GRDDL, Microdata and ODP**. 2011.

OLIVEIRA, D. M.; MENEGAZZO, C. T.; CLARO, D. B. **Uma Análise Conceitual das Linguagens Semânticas de serviços *Web* focando nas Composições:** Comparação entre OWL-S, WSMO e SAWSDL. UFBA, p.1-11, 2009.

PARASHAR, M.; HARIRI, S. **Autonomic Computing:** Concepts, Infrastructure, and Applications**.** University of Arizona, Tucson, AZ, USA. ISBN 9780849393679, p.1-24, 2005.

PERERA, C.; ZASLAVSKY, A.; CHRISTEN, P.; GEORGAKOPOULOS, D. **Context aware computing for the *Internet of Things*:** A survey**.** IEEE Communications Surveys & Tutorials, v.16, p.414-454, 2014.

PINHEIRO, J. M. S. ***Web* Semântica:** Uma rede de conceitos. Cadernos UniFOA, v. 8, n. 09, p.1-52, 2009.

PSCHORR, J.; HENSON, C. A.; PATNi, H. K.; SHETH, A. P. **Sensor discovery on linked data**. Proceedings of the 7th Extended *Semantic Web* Conference, ESWC2010, Heraklion, Greece, p.1-16, 2010.

REUSING, T. **Comparison of operating systems tinyos and *Contiki***. Sens. Nodes-Oper. Netw. Appl, v.7, p.15-72, 2012.

SEHGAL, A. **Using the *Contiki Cooja* Simulator**. Computer Science, Jacobs University Bremen Campus Ring, v.1, p.1-7, 2013.

SERRANO, M.; FOGHLÚ, M.; DONNELLY, W. **Enabling Information Interoperability in the Future Internet**: beyond of a SOA Design Requirement. FIA Session Linked Data in the Future Internet, p.1-10, 2010.

SPORNY, M.; KELLOGG, G.; LANTHALER, M. **JSON-LD 1.0:** a JSON-based serialization for linked data. W3C RDF Working Group. *W3C Recommendation*, v.16, p.1-33, 2014.

SUN, C. **Application of RFID Technology for Logistics on Internet of Things**. AASRI Procedia, v.1, p.106-111, 2012.

SURE, Y.; ERDMANN, M.; ANGELE, J.; STAAB, S.; STUDER, R.; WENKE, D. **OntoEdit:** Collaborative ontology development for the *semantic web*. InInternational *Semantic Web* Conference, Springer Berlin Heidelberg, p.221-223, 2002.

STERRITT, R.; BUSTARD, D. **Autonomic Computing A Means of Achieving Dependability.** In Proceedings of IEEE International Conference on the Engineering of

Computer Based Systems (ECBS'03). Los Alamitos, CA, USA. IEEE Computer Society, p.247–251, 2003.

TOMBERG, V.; LAANPERE, M. **RDFa versus Microformats:** exploring the potencial for *semantic* interoperability. Mash-up Personal Learning Enviroments. SBC, v.506, p.102-109, 2013.

VILLAMOR, J. I. F.; FERNANDEZ, C. A. I.; AYESTARAN, M.G. **A vocabulary for the modelling of image search microservices**. P.1-8, 2010.

VITERBO, J. **Descoberta de Serviços em Ambientes de Computação Ubíqua**. PUC – Rio de Janeiro, v.8, p.11-13, 2015.

ZANELLA, A.; BUI, N.; CASTELLANI, A.; VANGELISTA, L.; ZORZI, M. ***Internet* of Things for smart cities,** IEEE *Internet* of Things Journal, v.1, p.22-32, 2014.

ZENG, D.; GUO, S.; CHENG Z. **The Web of Things:** A Survey. Journal of Communications, v.6, n.6, p.424-438, 2011.

WALMSLEY, R.; BRUTZMAN, D.; CARLSON, J. **A JSON encoding for X3D**. In: Proceedings of the 21st International Conference on *Web*3D Technology. ACM, 2016, p. 27-32, 2016.

WANG, W.; et al. **A comprehensive ontology for knowledge representation in the *Internet* of Things.** IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications. IEEE, p.1793-1798, 2012.

WHITEHEAD, J. **Microformats:** The next (Small) thing on the semântica *Web*?**.** IEEE INTERNET COMPUTING, v.10, p.68-75, 2006.

W3C. **OWL-S:** *Semantic Markup* for *Web* Services. W3C Member Submission, v.22, p.1-38, 2004.

APÊNDICE A

Apêndice A.1 – CÓDIGO APLICAÇÃO MELHOR PREÇO

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "contiki.h"
#include "contiki-net.h"
#include "lib/random.h"
#include "sys/ctimer.h"
#include "sys/etimer.h"
#include "net/uip.h"
#include "net/uip-ds6.h"
#include "simple-udp.h"
#if !UIP_CONF_IPV6_RPL && !defined (CONTIKI_TARGET_MINIMAL_NET) &&
!defined (CONTIKI_TARGET_NATIVE)
#warning "Compiling with static routing!"
#include "static-routing.h"
#endif


#include "erbium.h"
#if WITH_COAP == 3
#include "er-coap-03.h"
#elif WITH_COAP == 7
#include "er-coap-07.h"
#else
#warning "Erbium example without CoAP-specifc functionality"
#endif /* CoAP-specific example */

#if WITH_COAP == 3
#include "er-coap-03-engine.h"
#elif WITH_COAP == 6
#include "er-coap-06-engine.h"
#elif WITH_COAP == 7
#include "er-coap-07-engine.h"
#else
#error "CoAP version defined by WITH_COAP not implemented"
#endif

#define LOCAL_PORT      UIP_HTONS(COAP_DEFAULT_PORT+1)
#define REMOTE_PORT     UIP_HTONS(COAP_DEFAULT_PORT)
#define MAX_MERCADOS 5
#define MY_IP "65152,0,0,0,530,29698,2,514,"
#define REST_PUT_BROADCAST(ip, url, handler, corpo, atr)
coap_init_message (requisicao, COAP_TYPE_CON, COAP_PUT, 0); \
coap_set_header_uri_path (requisicao, url); \
coap_set_header_uri_query (requisicao, atr); \
coap_set_header_content_type (requisicao, APPLICATION_XML); \
```

```
coap_set_payload(requisicao, (uint8_t *) corpo, sizeof (corpo)-1);
\
COAP_BLOCKING_REQUEST (ip, REMOTE_PORT, requisicao, handler)
#define REST_GET(ip, url, handler, corpo, atr)
oap_init_message(request, COAP_TYPE_CON, COAP_GET, 0); \
coap_set_header_uri_path(request, url); \
coap_set_header_uri_query(request, atr); \
coap_set_header_content_type(request, APPLICATION_XML); \
coap_set_payload(request, (uint8_t *)corpo, sizeof(corpo)-1); \
COAP_BLOCKING_REQUEST(ip, REMOTE_PORT, request, handler)
#define LER_IP_PARTE(str) i_l = 0; \
while (str [comeco] != ',') \
                          { \
                            ip_parte [i_l++] = str[comeco++]; \
                          } \
                          ip_parte [i_l] = 0; \
                          comeco++

#define LER_IP(str, inic) int comeco = inic, j; \
                          LER_IP_PARTE(str); for(j=0; j<=i_l; j++)
ip_a[j] = ip_parte[j]; \
                          LER_IP_PARTE(str); for(j=0; j<=i_l; j++)
ip_b[j] = ip_parte[j]; \
                          LER_IP_PARTE(str); for(j=0; j<=i_l; j++)
ip_c[j] = ip_parte[j]; \
                          LER_IP_PARTE(str); for(j=0; j<=i_l; j++)
ip_d[j] = ip_parte[j]; \
                          LER_IP_PARTE(str); for(j=0; j<=i_l; j++)
ip_e[j] = ip_parte[j]; \
                          LER_IP_PARTE(str); for(j=0; j<=i_l; j++)
ip_f[j] = ip_parte[j]; \
                          LER_IP_PARTE(str); for(j=0; j<=i_l; j++)
ip_g[j] = ip_parte[j]; \
                          LER_IP_PARTE(str); for(j=0; j<=i_l; j++)
ip_h[j] = ip_parte[j]; \
                          i_l = comeco
struct TYPE {
    int cod;
    uip_ipaddr_t ip;
};
size_t tamanho = 0;

struct TYPE
  mercados [MAX_MERCADOS];

int
  aux,
  i,
  j,
  codigo;
```

```
static int
  quantidadeMercados;
uint8_t
  *conteudo;
static coap_packet_t
  requisicao [1];
struct etimer
  tempo;
static char
  msg [100]; // O tamanho maximo dos dados e de 64 bytes!!!
char
  ip_a [10],
  ip_b [10],
  ip_c [10],
  ip_d [10],
  ip_e [10],
  ip_f [10],
  ip_g [10],
  ip_h [10],
  ip_parte [10];
static uip_ipaddr_t
  addr;

// Monta o conteudo da resposta: o vocabulario com os servicos
contidos em menor preco
// Comprimento da mensagem de resposta: 1305 bytes
void montaResposta (char *resposta)
{
  resposta [0] = '\0';
  strcat (resposta,
"Servico:servicosdisponiveis{\"@context\":{\"ServicoIoT\":\"Discove
ry\",");
    strcat (resposta,
"\"dc\":\"http://purl.org/dc/terms/\",\"cc\":\"http://creativecommo
ns.org/ns#\",");
    strcat (resposta, "\"title\":
\"Discovery\",\"description\":\"listas os servicos
disponiveis\",");
    strcat (resposta,
"\"entrypoint\":{\"@id\":\"ServicoIoT:Entrypoint\",\"@type\":
\"@id\"},");
    strcat (resposta,
"\"CadastraMercado\":\"ServicoIoT:cadastraMercado\",\"CompraProduto
\":\"ServicoIoT:compraProduto\",");
  strcat (resposta, "\"@id\":\"Vocabulario para descoberta de
servicos de menor preco\",\"@type\":\"Json:vocabulario
semantico\",");
  strcat (resposta, "\"label\":\"Adaptacao de vocabulario generico
de descoberta de servico\",\"preferredprefix\":\"ServicoIoT\",");
```

```
  strcat (resposta, "\"dc:description\":\"Vocabulario de teste para
o servidor de menor preco\",");

  strcat (resposta,
"\"defines\":{{\"@id\":\"ServicoIoT:Entrypoint\",\"label\":\"Discov
ery\",");

  strcat (resposta, "\"comment\":\"link para a descoberta de
servico\",\"method\":\"GET\"\"return\":\"servicoIoT:Vocabulario\"},
");
  strcat (resposta,
"{\"@id\":\"ServicoIoT:cadastraMercado\",\"label\":\"cadastraMercad
o\",\"comment\":\"Cadastra mercado e produto\",");

  strcat (resposta,
"\"method\":\"PUT\",\"parameters\":{\"codigomercado\":\"ServicoIoT:
propriedadecodigo\",");

  strcat (resposta,
"\"codigoProduto\":\"ServicoIoT:propriedadecodigo\",\"quantidade\":
\"ServicoIoT:quantidade\",\"valor\":\"servicoIoT:valor\"}},");

  strcat (resposta,
"{\"@id\":\"ServicoIoT:compraProduto\",\"label\":\"compraProduto\",
\"comment\":\"Compra produtos\",");

  strcat (resposta,
"\"method\":\"GET\",\"parameters\":{\"codigoProduto\":\"ServicoIoT:
propriedadecodigo\",\"quantidade\":\"ServicoIoT:quantidade\",}");

  strcat (resposta, "\"return\":\"servicoIoT:retorno\",}}}");
}
// Recurso para a descoberta dos servicos contidos no servidor
RESOURCE (discovery, METHOD_GET, "discovery", "");

// Funcao para gerir as requisicoes de GET de descoberta de servico
vindas de geladeiras ou supermercados
// A responsabilidade desta funcao e enviar o vocabulario com as
definicoes de servico como resposta
void discovery_handler (void* request, void* response, uint8_t
*buffer, uint16_t preferred_size, int32_t *offset)
{
  char
    resposta [2048]; // String contendo o vocabulario com os
servicos contidos em menor preco

  printf("[MELHOR PRECO] (discovery_handler) Descoberta de servico
recebida.\n");
```

```
 printf("[MELHOR PRECO] (discovery_handler) Tamanho maximo em bytes
da resposta: %d.\n", preferred_size);

  // Monta a resposta na respectiva string
  montaResposta (resposta);

  printf("[MELHOR PRECO] (discovery_handler) Resposta enviada:
%s.\n", resposta);

  printf ("[MELHOR PRECO] (discovery_handler) Tamanho da resposta
em bytes: %d\n", strlen (resposta));

  // A resposta e copiada para o buffer com o tamanho maximo
  // Caso tente-se usar o tamanho real do vocabulario (1305 bytes),
ocorre um erro
  // em tempo de simulacao, por isso a linha com strlen (resposta)
esta comentada
  //memcpy(buffer, resposta, strlen (resposta));
  memcpy(buffer, resposta, 64);

  printf("[MELHOR PRECO] (discovery_handler) Resposta enviada -
buffer: %s.\n", buffer);

  // Informa que o conteudo e JSON
  REST.set_header_content_type (response, APPLICATION_JSON);

  // Envia o vocabulario como resposta ao cliente que solicitou a
descoberta de servico
  // Tentar enviar os 1305 bytes da resposta gera um erro em tempo
de simulacao
  // O maximo de dados que pode ser enviado sao 64 bytes
  //REST.set_response_payload (response, buffer, strlen
(resposta));
  REST.set_response_payload (response, buffer, 64);

}


// Recebe anuncio de servico de mercado
// Cada mercado envia endereco IPv6 e codigo
// Isso e armazenado em um vetor de mercados
RESOURCE (discovery_mercado, METHOD_PUT,
"server/mercado/discovery", "title=\"cod=...\";rt=\"Discovery\"");

void discovery_mercado_handler (void* request, void* response,
uint8_t *buffer, uint16_t preferred_size, int32_t *offset)
{
  printf("[MELHOR PRECO] (discovery_mercado_handler) Recebeu
anuncio de servico de mercado.\n");
```

```
  if (quantidadeMercados < MAX_MERCADOS)
  {
      int
        tamanho = coap_get_payload (request, &conteudo);

      //printf ("[MELHOR PRECO] (discovery_mercado_handler)
Conteudo da requisicao: %s\n", conteudo);

      if (conteudo)
      {
       memcpy (msg, conteudo, tamanho);

       msg [tamanho] = 0;

       //printf ("[MELHOR PRECO] (discovery_mercado_handler)
Conteudo da requisicao: %s\n", conteudo);
      }

      const char
        *queryString = NULL;

      codigo = -1;

      if (coap_get_query_variable(request, "cod", &queryString))
      {
        char
          tempCodigo [2];

       tempCodigo [0] = queryString[0];

       tempCodigo [2] = '\0';

       codigo = atoi (tempCodigo);

       //printf ("[MELHOR PRECO] (discovery_mercado) Conteudo de
queryString: %s\n", queryString);
      }

      int
        found,
        temp,
        i_l;

      LER_IP(msg,0);

      found = 0;
      for (temp = 0; temp < quantidadeMercados; ++ temp)
        if (codigo == mercados [temp].cod)
          found = 1;
```

```
      // So insere um mercado se ele ainda nao existir
      if (found == 0)
      {
        // Insere um mercado no vetor
        uip_ip6addr (&mercados [quantidadeMercados].ip, atoi
(ip_a), atoi (ip_b), atoi (ip_c), atoi (ip_d), atoi (ip_e), atoi
(ip_f), atoi (ip_g), atoi (ip_h));

        mercados [quantidadeMercados++].cod = codigo;

        printf("[MELHOR PRECO] (discovery_mercado) Inseriu o
mercado de codigo %d\n", codigo);

        coap_set_payload (response, "1", 1); // Envia a resposta 0
para informar que a inclusao do mercado ocorreu
      }
      else
      {
        coap_set_payload (response, "0", 1); // Envia a resposta 0
para informar que a inclusao do mercado nao ocorreu
      }
  }
}

// Recurso para receber requisicoes de compra das geladeiras
RESOURCE (compra, METHOD_GET, "server/compra",
"title=\"Compra\";rt=\"Text\"");

void compra_handler (void* request, void* response, uint8_t
*buffer, uint16_t preferred_size, int32_t *offset)
{
  printf("[MELHOR PRECO] (compra_handler) Solicitacao de compra
recebida.\n");

  int tamanho = coap_get_payload (request, &conteudo);

  if (conteudo)
  {
    memcpy (msg, conteudo, tamanho);

    msg [tamanho] = 0;

    //printf ("[MELHOR PRECO] (compra_handler) Conteudo da compra:
%s",msg);
  }

  char
    *item = strstr (msg, "item");

  if (item != NULL)
```

```
  {
    printf ("[MELHOR PRECO] (compra_handler) Conteudo: %s\n", msg);

    memcpy (buffer, "01", 2);
  }
  else
  {
   memcpy (buffer, "-1", 2);
  }

  REST.set_response_payload (response, buffer, 2);
}


PROCESS (preco, "Menor Preco");

AUTOSTART_PROCESSES (&preco);

PROCESS_THREAD (preco, ev, data)
{
  PROCESS_BEGIN ();

  printf("[MELHOR PRECO] (principal) Melhor preco iniciado.\n");

  rest_init_engine();

  rest_activate_resource(&resource_discovery);

  rest_activate_resource(&resource_compra);

  rest_activate_resource(&resource_discovery_mercado);

  quantidadeMercados = 0;

  while (1)
  {
    PROCESS_WAIT_EVENT ();

  }

  PROCESS_END ();
}
```

Apêndice A.2 – CÓDIGO APLICAÇÃO SUPERMERCADO

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "contiki.h"
#include "contiki-net.h"
#include "node-id.h"

#include "lib/random.h"
#include "sys/ctimer.h"
#include "sys/etimer.h"
#include "net/uip.h"
#include "net/uip-ds6.h"
#include "simple-udp.h"


#if !UIP_CONF_IPV6_RPL && !defined (CONTIKI_TARGET_MINIMAL_NET) &&
!defined (CONTIKI_TARGET_NATIVE)
#warning "Compiling with static routing!"
#include "static-routing.h"
#endif

#include "erbium.h"

#include "dev/button-sensor.h"

#if WITH_COAP == 3
#include "er-coap-03-engine.h"
#elif WITH_COAP == 6
#include "er-coap-06-engine.h"
#elif WITH_COAP == 7
#include "er-coap-07-engine.h"
#else
#error "CoAP version defined by WITH_COAP not implemented"
#endif

#define MY_IP "65152,0,0,0,530,29700,4,1028,"

#define SERVER_NODE(ipaddr)   uip_ip6addr(&ipaddr, 0xfe80, 0, 0, 0,
0x0212, 0x7402, 0x0002, 0x0202) /* cooja2 */

#define LOCAL_PORT      UIP_HTONS(COAP_DEFAULT_PORT+1)
#define REMOTE_PORT UIP_HTONS(COAP_DEFAULT_PORT)

#define REST_PUT_BROADCAST(ip,url, handler, corpo, atr)
coap_init_message (requisicao, COAP_TYPE_CON, COAP_PUT, 0); \

coap_set_header_uri_path (requisicao, url); \
```

```
coap_set_header_content_type (requisicao, APPLICATION_XML);\

coap_set_header_uri_query (requisicao, atr);\

coap_set_payload (requisicao, (uint8_t *) corpo, sizeof (corpo)-1);
/* coap_set_payload (requisicao, (char *) corpo, strlen (corpo));
*/ \

COAP_BLOCKING_REQUEST (ip, REMOTE_PORT, requisicao, handler)

#define REST_GET(ip, url, handler, corpo, atr)
coap_init_message(request, COAP_TYPE_CON, COAP_GET, 0); \

coap_set_header_uri_path(request, url); \

coap_set_header_uri_query(request, atr); \

coap_set_header_content_type(request, APPLICATION_XML); \

coap_set_payload(request, (uint8_t *)corpo, sizeof(corpo)-1); \

COAP_BLOCKING_REQUEST(ip, REMOTE_PORT, request, handler)

int
  servicoDescoberto,
  mercadoRecebido;

size_t
  tamanho = 0;

uint8_t
  *conteudo;

char
  msg [100],
  codigoSupermercado [100]; // O tamanho maximo dos dados e de 64
bytes!!!

static coap_packet_t
  request [1],
  requisicao [1];

struct etimer
  tempo;

static uip_ipaddr_t
  addr;
```

```
// Manipula as respostas do servidor de melhor preco para testar o
vocabulario
void discovery_vocabulario_handler (void *response)
{
  printf ("[GELADEIRA] (discovery_vocabulario) Recebeu resposta de
servidor de melhor preco.\n");

  uint16_t
      payload_len = 0;

    uint8_t*
      payload = NULL;

    char
      temp [100];

    payload_len = coap_get_payload(response, &payload);

      if (payload)
      {
        memcpy(temp, payload, payload_len);

        temp[payload_len] = 0;

        printf("[MERCADO] (discovery_vocabulario) payload: %s\n",
temp);
      }

  // Se receber resposta do servidor de melhor preco, pode
modificar o flag
  servicoDescoberto = 1;

}

void broadcast_handler (void *response)
{
  uint16_t
    payload_len = 0;

  uint8_t*
    payload = NULL;

  char
    temp [100];

  payload_len = coap_get_payload(response, &payload);

    if (payload)
    {
      memcpy(temp, payload, payload_len);
```

```
      temp[payload_len] = 0;

      printf("[MERCADO] (broadcast_handler) payload: %s\n", temp);
    }


  // Nao importa o conteudo da resposta!!!
  // Se chegar uma resposta, o mercado ja foi inserido em melhor
preco.
  // Essa resposta faz com que o mercado pare de enviar anuncio de
servico
  // ao servidor de menor preco
  mercadoRecebido = 1;
}

RESOURCE (consulta, METHOD_GET, "mercado/consulta",
"title=\"consulta?item=...\";rt=\"Text\"");

void consulta_handler (void* request, void* response, uint8_t
*buffer, uint16_t preferred_size, int32_t *offset)
{
  const char
    *codigo_produto = NULL;

  const char
    *message = "<prd><cod>";

  if (REST.get_query_variable (request, "item", &codigo_produto))
  {
     strcat((char *)message, codigo_produto);

     strcat((char *)message, "</cod><pr>37</pr></prd>");

     memcpy(buffer, message, strlen(message));
  }

  REST.set_header_content_type (response, APPLICATION_XML);

  REST.set_response_payload (response, buffer, strlen(message));
}

PROCESS (mercado, "Mercado");

AUTOSTART_PROCESSES (&mercado);

PROCESS_THREAD (mercado, ev, data)
{

  PROCESS_BEGIN ();
```

```
  // Flag de mercado recebido
  mercadoRecebido = 0;

  // Flag de servico descoberto (teste do vocabulario)
  servicoDescoberto = 0;

  // A cada 30 segundos realiza uma compra
  etimer_set (&tempo, 30 * CLOCK_SECOND);

  printf ("[MERCADO] (principal) Supermercado na rede.\n");

  // Define o codigo do supermercado como sendo o numero do no
atribuido pelo contiki
  sprintf (codigoSupermercado, "?cod=%d", node_id);

  //printf ("[MERCADO] (principal) node-id: %s.\n",
codigoSupermercado);

  // Inicia a engine de REST
  rest_init_engine ();

  // Ativa o recurso de consulta de preco via REST
  rest_activate_resource (&resource_consulta);

  // Atribui o endereco IPv6 do servidor de melhor preco
  SERVER_NODE (addr);

  while (1)
  {
    PROCESS_WAIT_EVENT ();

    if (etimer_expired (&tempo))
    {
      // Se o servico nao estiver descoberto, envia GET para
discovery do servidor de melhor preco
            if (!servicoDescoberto)
            {
              // Exibe mensagem de consulta de vocabulario
              printf ("[MERCADO] (principal) Solicitando servicos -
teste de vocabulario.\n");

              // Faz GET no servidor de menor preco para testar o
vocabulario
              REST_GET(&addr, "discovery",
discovery_vocabulario_handler, "SERVICES", " ");

            }
```

```
        if (!mercadoRecebido && servicoDescoberto) // So envia ao
menor preco se ele nao enviar a resposta com reconhecimento
        {
          printf ("[MERCADO] (principal) envia IP e codigo a menor
preco.\n");

          REST_PUT_BROADCAST (&addr, "server/mercado/discovery",
broadcast_handler, MY_IP, codigoSupermercado);
        }

       etimer_reset (&tempo);
    }
  }

  PROCESS_END ();
}

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "contiki.h"
#include "contiki-net.h"
#include "node-id.h"

#include "lib/random.h"
#include "sys/ctimer.h"
#include "sys/etimer.h"
#include "net/uip.h"
#include "net/uip-ds6.h"
#include "simple-udp.h"


#if !UIP_CONF_IPV6_RPL && !defined (CONTIKI_TARGET_MINIMAL_NET) &&
!defined (CONTIKI_TARGET_NATIVE)
#warning "Compiling with static routing!"
#include "static-routing.h"
#endif

#include "erbium.h"

#include "dev/button-sensor.h"

#if WITH_COAP == 3
#include "er-coap-03-engine.h"
#elif WITH_COAP == 6
#include "er-coap-06-engine.h"
#elif WITH_COAP == 7
#include "er-coap-07-engine.h"
#else
#error "CoAP version defined by WITH_COAP not implemented"
```

```
#endif

#define MY_IP "65152,0,0,0,530,29700,4,1028,"

#define SERVER_NODE(ipaddr)   uip_ip6addr(&ipaddr, 0xfe80, 0, 0, 0,
0x0212, 0x7402, 0x0002, 0x0202) /* cooja2 */

#define LOCAL_PORT      UIP_HTONS(COAP_DEFAULT_PORT+1)
#define REMOTE_PORT UIP_HTONS(COAP_DEFAULT_PORT)

#define REST_PUT_BROADCAST(ip,url, handler, corpo, atr)
coap_init_message (requisicao, COAP_TYPE_CON, COAP_PUT, 0); \

coap_set_header_uri_path (requisicao, url); \

coap_set_header_content_type (requisicao, APPLICATION_XML);\

coap_set_header_uri_query (requisicao, atr);\

coap_set_payload (requisicao, (uint8_t *) corpo, sizeof (corpo)-1);
/* coap_set_payload (requisicao, (char *) corpo, strlen (corpo));
*/ \

COAP_BLOCKING_REQUEST (ip, REMOTE_PORT, requisicao, handler)

#define REST_GET(ip, url, handler, corpo, atr)
coap_init_message(request, COAP_TYPE_CON, COAP_GET, 0); \

coap_set_header_uri_path(request, url); \

coap_set_header_uri_query(request, atr); \

coap_set_header_content_type(request, APPLICATION_XML); \

coap_set_payload(request, (uint8_t *)corpo, sizeof(corpo)-1); \

COAP_BLOCKING_REQUEST(ip, REMOTE_PORT, request, handler)

int
  servicoDescoberto,
  mercadoRecebido;

size_t
  tamanho = 0;

uint8_t
  *conteudo;

char
  msg [100],
```

```
  codigoSupermercado [100]; // O tamanho maximo dos dados e de 64
bytes!!!

static coap_packet_t
  request [1],
  requisicao [1];

struct etimer
  tempo;

static uip_ipaddr_t
  addr;

// Manipula as respostas do servidor de melhor preco para testar o
vocabulario
void discovery_vocabulario_handler (void *response)
{
  printf ("[GELADEIRA] (discovery_vocabulario) Recebeu resposta de
servidor de melhor preco.\n");

  uint16_t
      payload_len = 0;

    uint8_t*
      payload = NULL;

    char
      temp [100];

    payload_len = coap_get_payload(response, &payload);

      if (payload)
      {
        memcpy(temp, payload, payload_len);

        temp[payload_len] = 0;

        printf("[MERCADO] (discovery_vocabulario) payload: %s\n",
temp);
      }

  // Se receber resposta do servidor de melhor preco, pode
modificar o flag
  servicoDescoberto = 1;

}

void broadcast_handler (void *response)
{
  uint16_t
```

```
    payload_len = 0;

  uint8_t*
    payload = NULL;

  char
    temp [100];

  payload_len = coap_get_payload(response, &payload);

    if (payload)
    {
      memcpy(temp, payload, payload_len);

      temp[payload_len] = 0;

      printf("[MERCADO] (broadcast_handler) payload: %s\n", temp);
    }


  // Nao importa o conteudo da resposta!!!
  // Se chegar uma resposta, o mercado ja foi inserido em melhor
preco.
  // Essa resposta faz com que o mercado pare de enviar anuncio de
servico
  // ao servidor de menor preco
  mercadoRecebido = 1;
}

RESOURCE (consulta, METHOD_GET, "mercado/consulta",
"title=\"consulta?item=...\";rt=\"Text\"");

void consulta_handler (void* request, void* response, uint8_t
*buffer, uint16_t preferred_size, int32_t *offset)
{
  const char
    *codigo_produto = NULL;

  const char
    *message = "<prd><cod>";

  if (REST.get_query_variable (request, "item", &codigo_produto))
  {
      strcat((char *)message, codigo_produto);

      strcat((char *)message, "</cod><pr>37</pr></prd>");

      memcpy(buffer, message, strlen(message));
  }
```

```
  REST.set_header_content_type (response, APPLICATION_XML);

  REST.set_response_payload (response, buffer, strlen(message));
}

PROCESS (mercado, "Mercado");

AUTOSTART_PROCESSES (&mercado);

PROCESS_THREAD (mercado, ev, data)
{

  PROCESS_BEGIN ();

  // Flag de mercado recebido
  mercadoRecebido = 0;

  // Flag de servico descoberto (teste do vocabulario)
  servicoDescoberto = 0;

  // A cada 30 segundos realiza uma compra
  etimer_set (&tempo, 30 * CLOCK_SECOND);

  printf ("[MERCADO] (principal) Supermercado na rede.\n");

  // Define o codigo do supermercado como sendo o numero do no
atribuido pelo contiki
  sprintf (codigoSupermercado, "?cod=%d", node_id);

  //printf ("[MERCADO] (principal) node-id: %s.\n",
codigoSupermercado);

  // Inicia a engine de REST
  rest_init_engine ();

  // Ativa o recurso de consulta de preco via REST
  rest_activate_resource (&resource_consulta);

  // Atribui o endereco IPv6 do servidor de melhor preco
  SERVER_NODE (addr);

  while (1)
  {
    PROCESS_WAIT_EVENT ();

    if (etimer_expired (&tempo))
    {
      // Se o servico nao estiver descoberto, envia GET para
discovery do servidor de melhor preco
            if (!servicoDescoberto)
```

```
            {
              // Exibe mensagem de consulta de vocabulario
              printf ("[MERCADO] (principal) Solicitando servicos -
teste de vocabulario.\n");

              // Faz GET no servidor de menor preco para testar o
vocabulario
              REST_GET(&addr, "discovery",
discovery_vocabulario_handler, "SERVICES", " ");

            }


        if (!mercadoRecebido && servicoDescoberto) // So envia ao
menor preco se ele nao enviar a resposta com reconhecimento
        {
          printf ("[MERCADO] (principal) envia IP e codigo a menor
preco.\n");

          REST_PUT_BROADCAST (&addr, "server/mercado/discovery",
broadcast_handler, MY_IP, codigoSupermercado);
        }

       etimer_reset (&tempo);
    }
  }

  PROCESS_END ();
}
```

Apêndice A.3– CÓDIGO APLICAÇÃO GELADEIRA

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "contiki.h"
#include "contiki-net.h"

#include "lib/random.h"
#include "sys/ctimer.h"
#include "sys/etimer.h"
#include "net/uip.h"
#include "net/uip-ds6.h"
#include "simple-udp.h"


//#include "powertrace.h"

#if !UIP_CONF_IPV6_RPL && !defined (CONTIKI_TARGET_MINIMAL_NET) &&
!defined (CONTIKI_TARGET_NATIVE)
#warning "Compiling with static routing!"
#include "static-routing.h"
#endif

#include "erbium.h"

#include "dev/button-sensor.h"

#if WITH_COAP == 3
#include "er-coap-03-engine.h"
#elif WITH_COAP == 6
#include "er-coap-06-engine.h"
#elif WITH_COAP == 7
#include "er-coap-07-engine.h"
#else
#error "CoAP version defined by WITH_COAP not implemented"
#endif

#define SERVER_NODE(ipaddr)   uip_ip6addr(&ipaddr, 0xfe80, 0, 0, 0,
0x0212, 0x7402, 0x0002, 0x0202) /* cooja2 */

#define LOCAL_PORT      UIP_HTONS(COAP_DEFAULT_PORT+1)
#define REMOTE_PORT     UIP_HTONS(COAP_DEFAULT_PORT)

#define MY_IP "65152,0,0,0,530,29699,3,771,"

#define LER_IP_PARTE(str)  i_l = 0; \
                           while (str [comeco] != ',') \
                           { \
                             ip_parte [i_l++] = str[comeco++]; \
```

```
                            } \
                            ip_parte [i_l] = 0; \
                            comeco++

#define LER_IP(str, inic) int comeco = inic, j, i_l; \
                            LER_IP_PARTE(str); for(j=0; j<=i_l; j++)
ip_a[j] = ip_parte[j]; \
                            LER_IP_PARTE(str); for(j=0; j<=i_l; j++)
ip_b[j] = ip_parte[j]; \
                            LER_IP_PARTE(str); for(j=0; j<=i_l; j++)
ip_c[j] = ip_parte[j]; \
                            LER_IP_PARTE(str); for(j=0; j<=i_l; j++)
ip_d[j] = ip_parte[j]; \
                            LER_IP_PARTE(str); for(j=0; j<=i_l; j++)
ip_e[j] = ip_parte[j]; \
                            LER_IP_PARTE(str); for(j=0; j<=i_l; j++)
ip_f[j] = ip_parte[j]; \
                            LER_IP_PARTE(str); for(j=0; j<=i_l; j++)
ip_g[j] = ip_parte[j]; \
                            LER_IP_PARTE(str); for(j=0; j<=i_l; j++)
ip_h[j] = ip_parte[j]; \
                            i_l = comeco

#define REST_PUT_BROADCAST(ip, url, handler, corpo, atr)
coap_init_message(request, COAP_TYPE_CON, COAP_PUT, 0); \

coap_set_header_uri_path(request, url); \

coap_set_header_uri_query(request, atr); \

coap_set_header_content_type(request, APPLICATION_XML); \

coap_set_payload(request, (uint8_t *)corpo, sizeof(corpo)-1); \

COAP_BLOCKING_REQUEST(ip, REMOTE_PORT, request, handler)

#define REST_GET(ip, url, handler, corpo, atr)
coap_init_message(requisicao, COAP_TYPE_CON, COAP_GET, 0); \

coap_set_header_uri_path(requisicao, url); \

coap_set_header_uri_query(requisicao, atr); \

coap_set_header_content_type(requisicao, APPLICATION_XML); \

coap_set_payload(requisicao, (uint8_t *)corpo, sizeof(corpo)-1); \

COAP_BLOCKING_REQUEST(ip, REMOTE_PORT, requisicao, handler)
```

```
//#define COMPRA
"<compra><cliente>900</cliente><item>112</item><quantidade>2</quant
idade></compra>"

#define COMPRA
"<cp><cli>900</cli><item>112</item><qtd>2</qtd></cp>"

static uip_ipaddr_t
  addr;

static struct etimer
  tempo;

static int
  servicoDescoberto,
  compraEfetuada;

uint8_t
  *conteudo;

char
  ip_a [10],
  ip_b [10],
  ip_c [10],
  ip_d [10],
  ip_e [10],
  ip_f [10],
  ip_g [10],
  ip_h [10],
  ip_parte [10];

static coap_packet_t
  requisicao [1],
  request [1];

// Manipula as respostas do servidor de melhor preco para testar o
vocabulario
void discovery_vocabulario_handler (void *response)
{
  printf ("[GELADEIRA] (discovery_vocabulario) Recebeu resposta de
servidor de melhor preco.\n");

  // Se receber resposta do servidor de melhor preco, pode
modificar o flag
  servicoDescoberto = 1;

}

// Funcao para manipular uma compra (resposta de um servidor de
melhor compra)
```

```
// O objetivo dessa funcao e efetuar uma compra. Isso e feito
quando um valor
// e retornado do servidor de melhor compra.
void compra_handler (void *response)
{
  uint8_t
      *chunk = NULL;
  uint16_t
      len = 0;
  char
    msg [100];

  //printf ("[GELADEIRA] (compra_handler) Entrou!!!\n");

   // Recupera os dados da mensagem recebida do servidor de melhor
compra
   // len e o comprimento da mensagem
   // response e a mensagem recebida
   // chunk sao os dados da mensagem recebida
   len = coap_get_payload(response, &chunk);

   // Se o conteudo da mensagem nao for vazio
   if (chunk)
   {
     // msg = chunk
     memcpy(msg, chunk, len);

     // Finaliza a string msg com \0
     msg[len] = 0;

     //printf ("[GELADEIRA] (compra_handler) Conteudo da compra:
%s\n", msg);

    // Converte o numero que esta em msg e armazena em len
    len = atoi(msg);

    //printf ("[GELADEIRA] (compra_handler) Valor da compra: %d\n",
len);

    // Se houver um valor armazenador em len significa que a compra
foi efetuada
    if (len)
    {
      printf ("[GELADEIRA] (compra_handler) Compra efetuada. Valor
total: %d\n",len);

      compraEfetuada = 1;
    }
    // Caso contrario houve falha na tentativa de compra
    else
```

```
    {
     printf ("[GELADEIRA] (compra_handler) Falha na tentativa de
compra.\n");

     compraEfetuada = 0;
    }
   }
}

// Definicoes da thread principal do no sensor
// Isso e padrao no Contiki
PROCESS (geladeira, "Geladeira");

AUTOSTART_PROCESSES (&geladeira);

// Thread principal do no sensor - TUDO INICIA AQUI EM QUALQUER NO
SENSOR
PROCESS_THREAD (geladeira, ev, data)
{
  // Inicio da thread principal
  PROCESS_BEGIN ();

     //powertrace_start (CLOCK_SECOND * 2); // Nao esta funcionando
porque nao cabe na memoria do no sensor

  // Inicia a engine REST
  rest_init_engine ();

  // Atribui zero a compra efetuada
  compraEfetuada = 0;

  // Atribui zero a servico descoberto
  servicoDescoberto = 0;

  // Inicia um timer para executar a cada 10 segundos
  etimer_set (&tempo, 10 * CLOCK_SECOND);

  // Atribui o endereco IPv6 do servidor de melhor preco
  SERVER_NODE (addr);

  // Laco principal da thread principal
  while (1) {

    // Aguarda por um evento, nesse caso receber um endereco IPv6
de um servidor em discover_handler
    PROCESS_WAIT_EVENT();

    // Se o tempo expirou - nesse caso se passou 10 segundos
    if (etimer_expired (&tempo))
    {
```

```
      // Se o servico nao estiver descoberto, envia GET para
discovery do servidor de melhor preco
      if (!servicoDescoberto)
      {
        // Exibe mensagem de consulta de vocabulario
        printf ("[GELADEIRA] (principal) Solicitando servicos -
teste de vocabulario.\n");

        // Faz GET no servidor de menor preco para testar o
vocabulario
        REST_GET(&addr, "discovery", discovery_vocabulario_handler,
"SERVICES", " ");

      }

      // Se a compra nao foi efetuada, efetua a mesma
      if (!compraEfetuada && servicoDescoberto)
      {
        // Exibe mensagem de solicitacao de compra
        printf ("[GELADEIRA] (principal) Solicitando compra.\n");

        // Faz um GET no mercado com o menor preco que foi
selecionado e usa a funcao compra_handler definida neste arquivo
        REST_GET(&addr, "server/compra", compra_handler, COMPRA, "
");
      }

      // Reseta o timer
      etimer_reset (&tempo);
    }

  }

  // Fim da thread principal
  PROCESS_END ();
}
```

APÊNDICE B

Apêndice B.1– REQUISIÇÃO DE SERVIÇOS EM JSON

Servico: ServicosDisponiveis


```
{
“@context”:{
        “ServicoIoT”:”Discovery”,
        “dc”:”http://purl.org/dc/terms/”,
        “cc”:”http://creativecommons.org/ns#”,
        “title”: “Discovery”,
        “description”:”listas os servicos disponiveis”,
“entrypoint”:{
        “@id”:”ServicoIoT:Entrypoint”,
        “@type”: “@id”
},
“CadastraMercado”:”ServicoIoT:cadastraMercado”,
“CompraProduto”:”ServicoIoT:compraProduto”,
“@id”:”Vocabulario para descoberta de servicos de menor preco”,
“@type”:”Json:vocabulario semantico”,
“label”:”Adaptacao de vocabulario generico de descoberta de servico”,
“preferredprefix”:”ServicoIoT”,
“dc:description”:”Vocabulario de teste para o servidor de menor preco”,
“defines”:{
        {
                “@id”:”ServicoIoT:Entrypoint”,
                “label”:”Discovery”,
                “comment”:”link para a descoberta de servico”,
                “method”:”GET”
                “return”:”servicoIoT:Vocabulario”
        },
        {
```

```
            “@id”:”ServicoIoT:cadastraMercado”,
            “label”:”cadastraMercado”,
            “comment”:”Cadastra mercado e produto”,
            “method”:”PUT”,
            “parameters”:{
                “codigomercado”:”ServicoIoT:propriedadecodigo”,
                “codigoProduto”:”ServicoIoT:propriedadecodigo”,
                “quantidade”:”ServicoIoT:quantidade”,
                “valor”:”servicoIoT:valor”
            }
        },
        {
            “@id”:”ServicoIoT:compraProduto”,
            “label”:”compraProduto”,
            “comment”:”Compra produtos”,
            “method”:”GET”,
            “parameters”:{
                “codigoProduto”:”ServicoIoT:propriedadecodigo”,
                “quantidade”:”ServicoIoT:quantidade”,
            }
            “return”:”servicoIoT:retorno”,
        }
}
}
```

ANEXOS

Proposal of a Standard Vocabulary for Services Discovery on the *Internet of Things*

Mayka de Souza Lima[1], Admilson de Ribamar L. Riberio[1], Edward David Moreno[1]
[1]Computing Department, Segipe Federal University, Aracaju, Brazil
maykalima@hotmail.com, admilson@ufs.br,edwdavid@gmail.com

Keywords: *Internet of Things*, *Web of Things*, vocabulary, *semantic*, mark-up, IoT, ontology.

Abstract: *Internet of Things* (IoT) is the paradigm that will dominate the computing world in the coming years. Thus, studies should be performed in order to ensure improvements in the search for technologies that can create new standardized *semantic* vocabularies to find services provided by IoT. The aim of this work is to understand the concepts of vocabularies of ontology as well as technologies that use such *semantic* vocabularies. Thus, it was concluded that the proposed vocabulary increase the chances of finding the IoT services, integrated with applications that work with many wireless sensor devices.

1 INTRODUCTION

With the exponential growth of *web* business investment, the *Internet* has become one of the main existing business channels nowadays. Soon the development of technologies to create *Internet* access platforms, faster and faster, efficient and cheap generated an unprecedented technological and social transformation.

Connect virtually any device to the *Internet* facilitates not only the design of intelligent buildings, but also makes possible remote monitoring of other devices from water energy and meters, environmental sensors, to medical implants. This interaction with other devices raises a paradigm that by 2025, according to forecast NIC (US National Intelligence Council) will dominate the computing world, called *Internet of Things* (IoT).

In short, the IoT is the diffuse presence of a variety *of Things* or objects around us, for example, RFID tags - Radio Frequency Identification, smart mobile phones, wireless sensor networks - WSN, between others, that communicate by exchanging many messages, even by simple sensors (Atzori, 2010).

Some projects are directed to metadata architectures production, but are made to take into account three aspects of interoperability: *semantic*, enabling the understanding of the meaning of each element of the described feature, along with the associations found in it, making sure the use of specific vocabularies, ontologies and metadata standards that are essential; syntactic, determining how metadata should be coded to the information transfer, using technology employed as the XML (eXtensible *Markup* Language) language; and structural, that specifies how resources are organized, along with the types involved and the possible values for each type (Perera, 2013).

Therefore, a document or a file where they are formally defined the relationships between concepts in the *semantic Web*, is called ontology. A taxonomy formed classes and subclasses of objects related to each other more a set of inference rules that can use language like DAML (DARPA mark-up language agent) developed as an inference based on RDF (Resource Description Framework).

Create or edit an ontology for *semantic* vocabulary can be accomplished through a tool called Protégé-2000. Widespread for the *semantic Web*, this tool allows to define vocabularies in different microformats as, for example RDF and RDF Schema. Highly customizable, allows the conceptual modeling of these languages, making it possible to create a standard vocabulary to discover services used by the IoT.

This work aims to propose a standard vocabulary, creating a necessary ontology for services used by the IoT, demonstrating its functioning and interaction with the human life reality.

The remainder of this paper is organized as follows. In section 2, will be shown on existing requirements nowadays and contribute to the creation of a *semantic* vocabulary. Section 3 describes the proposed *semantic* vocabulary that will be explained and possible tool that can be used in creating it. Finally, in section 4, the work related to the subject covered in this paper and its relation to the ontology and *semantic*. Conclusion and references are given in section 5.

2 *INTERNET OF THINGS* VOCABULARY REQUIREMENTS

The *semantic* vocabularies used by the *Internet of Things*, have basic requirements needed to extract the communication results with the various services or applications. These basic requirements are described in this paper, to name a few of the concepts that could be used to serve as the basis for creating a standard vocabulary, such as the language syntax, ontology and *semantic markup*.

2.1 Syntax

Through this feature the syntax of services provides interoperability, allowing the automation of the use of information based on their *semantic*s. Thus, to describe things, an unique to identify them, is necessary both in the current *Web* and *Semantic Web*, which are called features and identified using URIs (Uniform Resource Identifier).

The syntax description, refers to the XML standards (Extensible *Markup* Language), XML namespace and XML schema (W3C, 2014). The idea is to associate a namespace URI and even if a name appears in more than one space, must be unique in its namespace. These standards describe a class of objects called XML documents and the behavior of computer applications that process these documents.

Being an open standard (non-owner) and simple to read and write by *software* applications, a document in XML format makes it an excellent format to exchange data between different applications.

2.2 Ontology

Ontologies are key requirements to create a *semantic* vocabulary. They serve as support for the description, publication, discovery and *Web* service composition being: OWL-S (Ontology *Web* Language *Semantic*); OWL-Time, resources ontology and domains ontology.

The ontology language OWL-S is *semantic markup* and proposes an ontology language for *Web* services for general purpose, able to describe any type of *Web* services

regardless of the domain it belongs to the service.
OWL-Time defines a higher level of temporal concepts and contains a specification of the time theory required for *semantic Web* applications. Is used to describe temporal content *Web* pages, as well as describe properties and *web* services time constraints.
Ontologies resources proposes a description of *Web* services and their intention is to be used in conjunction with OWL-S. And the domain is used to describe of *Web* services in order to provide the *semantic*s necessary to enable automatic discovery and composition (Hachem, 2011).

2.3 *Semantic Markup*

The patterns of *semantic* annotations that emerged by the W3C, were: RDF, RDFa, Microformats and Microdata. The *semantic* annotation is the process by which includes *semantic* information or metadata and add machine-readable information to existing content.
Thus, if the marking is ready to find the most relevant result, the *semantic* annotation will add diversity and richness to the process. With Microformats, which are not a new language but an idea to solve the simplest problems of *semantic Web*, by today's standards and use in *semantic markup*, it is easy to use, with the most used formats, hCalendar (for publication events) and hCard (for people, companies and organizations in general).
The RDFS, XML-based language, considered a comprehensive language, allowing you to define ontologies with her. But, being a complex language is not feasible to use it to mark *Web* pages *semantic*ally. The RDFa language (RDF - in - attributes) is the standard *semantic* annotation that does not have the need of using new *markup* elements, being performed by corporate attributes on any element (Normay, 2013).
The microdata are considered the latest *markup* language and competitor of current standards of the *Semantic Web* annotations. Heavily influenced by microformats, adopting a range of attributes such as hCard, hCalendar and other microformats.

3 PROPOSAL FOR AN IOT VOCABULARY

The initial proposal will be creating a standard *semantic* vocabulary to obtain a better result to discover services used by the IoT.
Then, we use a tool called Protégé, to develop new *semantic* vocabulary to be tested with applications or devices that work in the IoT. Correlate the *Web* services more robust and effective manner, and to facilitate the search of users in the use of information in the applications with the IoT is the main purpose of this proposed paper.
With this platform, developed by the Stanford Medical Informatics research group at Stanford University medical school, will allow the ontology construction, namely the vocabulary to be used in the discovery of services in the IoT.
Based on the Java language, the Protégé is extensible and deals with a strong community of developers, academics and users, who use it to knowledge solutions in various areas such as biomedicine. Chosen as the *semantic* creation tool due to several factors: is a publisher of open source ontologies, supports editing *Web* client or via desktop client, access can be made via the *Web* or application installed on the machine, allowing the development of ontologies in various formats such as: OWL; RDF and XML Schema.
It also allows the use of existing vocabularies when creating the ontology

itself and the export of ontology to the desired format, in addition to having a graphical interface easier to handle, according (Horridge, 2014).

The graphical interface provides access to the menu and toolbar bar, and presently five viewing areas (views) which act as navigation modules and editing classes, attribute, forms, forums and search the database of knowledge, providing the data input and retrieval of information. The Protégé platform interface can be viewed in Figure 1. According to a pattern to be used can be constructed by the ontology vocabulary defined by this tool. Thus, describe the classes, subclasses, instance values and capable of interacting with the applications used in *Web* to IoT.

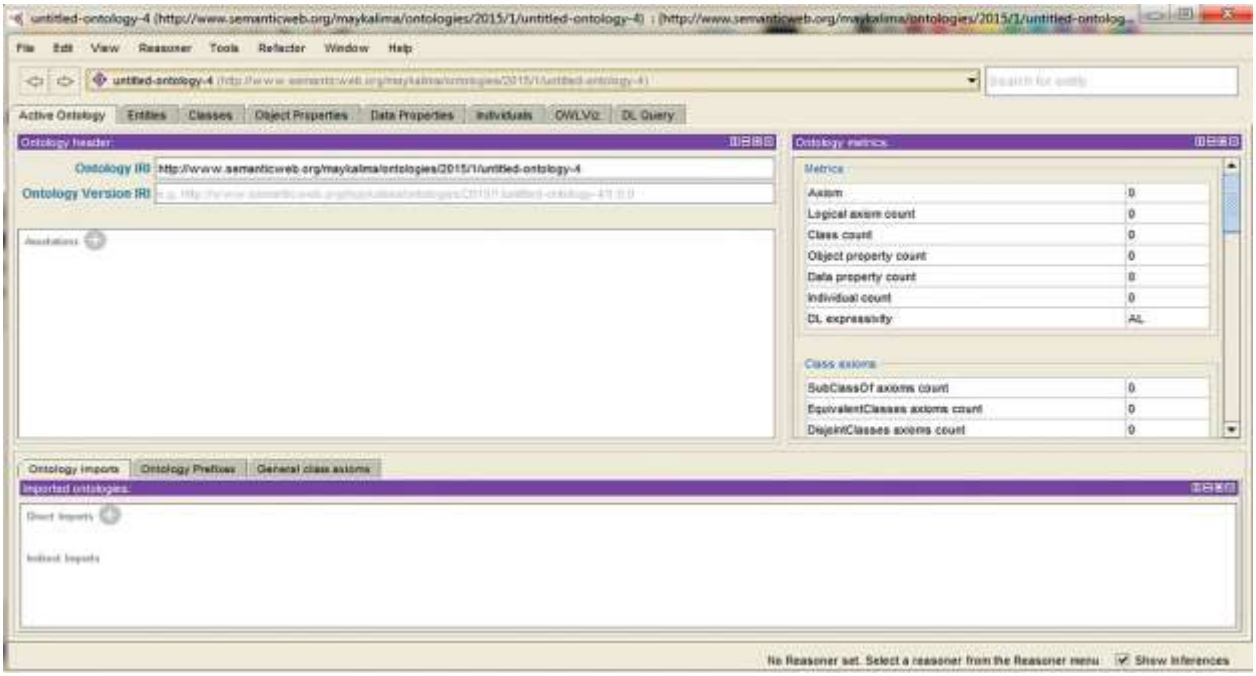

Figure 1: Protegé platform interface.

The vocabulary to be created should follow the standards previously defined in order to have more interaction with IoT applications and embedded devices. The whole structure to be mounted must have open-source systems, devices that are connected to the *Internet of Things*, and a *Web* interface to bring information services found by IoT.

Decrease the amount of human intervention plans in search of information as faster the *semantic* analysis and run the commands requested by users with better understanding are the metrics needed to get an effective result in the discovery of services connected to the IoT applications in various devices.

The *Web* Services Description Language (WSDL) can describe a service by specifying the location and describing the operations provided for him. All documents are in XML and provide enough information to interact with the *Web* service. Have as elements to define the service interface: types, XML schemas; messages describing details of the methods and their parameters; type of port that defines operations. Allowing its interaction with other systems that identify the service. A tool to be used for the discovery of services in WSDL will be the Lumina plugin. This ontology concepts associated with the input parameters, so that the search system according to see description to be made by the reference model (Menegazzo, 2009). Lumina is an eclipse plugin that allows to discover *Web* services based on *semantic* built. It has a graphical user interface and its repository services is UDDI (Universal Description, Discovery and Integration).

An example of the use of *semantic* vocabulary would be to use some client-side objects with a service by doing a proxy class that "mimic" the *web* service method calls. Working with the proxy instead of writing SOAP messages (Simple Object Access Protocol) directly. The proxy class manages the building, sending and receiving SOAP messages and OWL-S language is intended to enable the automation of discovery, invocation and *Web* Service composition through *semantic* descriptions. The axioms of OWL-S are defined in OWL and OWL-S has descriptions of classes, sub-class hierarchy and definitions of types and relationships between classes. The description of the functionality of a service is referenced by a profile of OWL-S ontology externally with the concepts defined in OWL ontology in the domain that owns the *Web* Service in question.

Analyzing the available work in this area, we can verify that applications involving IoT and *semantic* vocabularies in search of services need to get to an improvement in the syntax and *semantic*s of communication between them, creating new vocabularies that can interact with different devices connected to things.

4 RELATED WORK

In this section will be elucidated major systems that use *semantic* vocabulary for their services and their characteristics. These features are related to the requirements that have been seen before.

4.1 STEER

Computer program implemented to support mechanisms for service discovery, as well as tools and *semantic Web* services.

STEER compound (Task execution *Semantic* Editor) and PIPE (Pervasive Instance Provision Environment). A *Web* client with a Java interface classifies services based on their input data and *semantic* outputs. Through a GUI manager (Graphical User Interface) PIPE services are facilitated for the users so that in this way can perform *Web* services for common tasks. Using as ontology language, OWL, as a powerful modeling, naturally enough to describe many areas. (Masuoka, 2003).

4.2 Middleware IoT

Middleware architecture for it consists of three modules discovery, composition and estimation, and knowledge base. This application performs a request or a service by making detection, or a composition for estimation of the manipulated object. To solve a request, this module accesses the module (discovery) and makes the knowledge base by performing the creation of a composition of services (Zhexuan, 2010). The result is obtained by running this composition within the existing network.

4.3 Ontonym

Invasive system which has the need to interpret large amounts of data from many sources. Developed to address the temporal properties of the data context as a requirement. Having as a criterion to function using a set of ontologies to represent the fundamental concepts in pervasive computing (Stevenson, 2009).

The concepts that surround it to carry out a comparison with the ontology of pervasive computing are: time, using the ontology language OWL-Time; location, generated by a GPS; people, where personal and social networking information is collected; and the ontological sensors generating data.

The works related are limited in specific services for the use an ontology as OWL and OWL-S. The profile and the process model of these services specify *Web* service features that can be used by agents for *semantic* discovery, invocation and composition services.

With an ontology class, you can define how the service will be mapped from abstract definitions of the profile and the process for specific information to be made by means of messages exchanged between the user and application. Some of these applications, using traditional architectures for *Web* services, such as systems for wireless devices with the set of protocols UPnP (Universal Plug and Play) (Masuoka, 2003). However, OWL-S does not require the use of service oriented architectures and several studies reported in the literature using OWL-S based on other architectures.

Map the abstract elements of OWL-S of the evidence in WSDL (*Web* Services Description Language), and reuse all the existing infrastructure for *Web* services, maintaining that the *Semantic Web* should be an extension of the Current *Web*, leads to an advantage to locate *Web* services in the *Semantic Web* layers.

4.4 CoAP Framework

The COAP protocol (Constrained Application Protocol) devices for handling resources with a REST framework and is used in generic browsers for IoT. This protocol adopts HTTP resources (Hypertext Transfer Protocol) allowing an evolution of the *Web* from a simple document retrieval mechanism by GET REST methods, POST, and DELETE, integrating the URIs in the browser (Kovatsch, 2013).

From this interaction arise APIs (interactive programs) easy navigation of users on the *Web* through links, thus discovering new devices or services.

5 CONCLUSIONS

The vocabulary to be created will primary to provide as a benefit, the most effective discovery of the various services that are used by the *Internet of Things*. So, returning to the applications more concise information, making it easier to use between users and the various things interconnected to the *Internet*.

Once created, with the defined ontology language, this vocabulary will help the *semantic Web* of several applications that work-related RSSP networks and the *Internet of Things*. Therefore, the focus is the creation of standard vocabulary for the IoT, leaving all the attention to be directed to the main goal of the research.

For future work, there is the possibility to develop applications that use this *semantic* vocabulary demonstrating their interaction

with other devices available on the *Internet of Things* using different types of ontologies, syntax and *semantic*s.

REFERENCES

Atzori, L., Iera, A., Morabito, G, 2010. *The Internet Of Things: A Survey*. Italia. http://Www.Science.Smith.Edu/~Jcardell/Courses/Egr328/Readings/Iot%20survey.Pdf last accessed September 15, 2013.

Hachem, S., Teixeira, T., Issarny, V., 2011. Ontologies For The *Internet Of Things*. Inria Paris-Rocquencourt. Hal Id: Hal-00642193. https://Hal.Inria.Fr/Hal-00642193/Document last accessed January 01, 2015.

Horridge, M., 2014. Protégé. Manchester. http://owl.cs.manchester.ac.uk/publications/talks-and-tutorials/protg-owl-tutorial/ last accessed June 24, 2014.

Kovatsch, M., 2013. CoAP for the *Web of Things*: From Tiny Resource-constrained Devices to the *Web* th Internacional Workshop on the *Web of Things*. UbiComp'13 Adjunct. Zurich, Switzerland. http://www.vs.inf.ethz.ch/publ/papers/mkovatsc-2013-wot-copper.pdf last accessed March 15, 2015.

Masuoka, R., et al, 2013. Ontology – Enabled Pervasive Computing Applications. IEEE. University Of Maryland. http://masuoka.net/Ryusuke/papers/20030915-Task-Computing-IEEE-Intelligent-Systems-September-October-2003.pdf last accessed January 05, 2015.

Menegazzo, C., Oliveira, D., Claro, D. B, 2009. Uma análise conceitual das linguagens semânticas de serviços *Web*: Comparação entre OWL-S, WSMO e SAWSDL. UFBA – Universidade Federal da Bahia. http://homes.dcc.ufba.br/~dclaro/download/IADIS_CIAWI_2009_CAMERAREADY.pdf last accessed March 10, 2015.

Norway, F., 2013. *Semantic Markup* Report Microformats, Rdfa, Grddl, Microdata and Odp. Nce Tourism. http://www.vestforsk.no/filearchive/*semantic_markup*_report.pdf last accessed October 15, 2014.

Perera, C., Zaslaysky, A., Christen, P., 2013. Context Aware Computing For The *Internet Of Things*: A Survey. Ieee Communications Surveys & Tutorials Journal. Http://Arxiv.Org/Abs/1305.0982 last accessed May 18, 2014.

Stevenson, G., et al, 2009. Ontonym: A Collection Of Upper Ontologies or Developing Pervasive Systems. CIAO, Proceedings of the 1st Workshop on Context, Information and Ontologies. http://dl.acm.org/citation.cfm?id=1552271 last accessed December 15, 2014.

W3C, 2014. *OWL Web Ontology Language Overview*. Http://Www.W3.Org/Standards/*Semanticweb*/Ontology last accessed December 15, 2014.

Zhexuan, S., Cardenas, A., Masuoka, R., 2010. *Semantic* Middleware For The *Internet Of Things*. IEEE. Fujitsu Laboratories Of America, Inc. Http://Www.Flacp.Fujitsulabs.Com/~Cardenas/Papers/Iot2010.Pdf last accessed June 19, 2014.

Estudo da *Web* Semântica em Dispositivos Embarcados para *Internet* das Coisas (IoT)

Mayka de Souza Lima
Departamento de Computação (DCOMP) – Universidade Federal de Sergipe (UFS)
maykalima@hotmail.com
http://lattes.cnpq.br/4540312796926633
Wanderson Roger Azevedo Dias
Departamento de Computação (DCOMP) – Universidade Federal de Sergipe (UFS)
wanderson.dias@ifs.edu.br
http://lattes.cnpq.br/3742491905469878
Admilson de Ribamar Lima Ribeiro
Departamento de Computação (DCOMP) – Universidade Federal de Sergipe (UFS)
admilson@ufs.br
http://lattes.cnpq.br/0659299380542839
Edward David Moreno
Departamento de Computação (DCOMP) – Universidade Federal de Sergipe (UFS)
edwdavid@gmail.com
http://lattes.cnpq.br/8377190526783442

RESUMO

A *Web* semântica é uma nova maneira de atribuir um significado ao conteúdo da *Web* com um vocabulário. Atualmente existem várias pesquisas a respeito da construção de ontologias ou vocabulários que visam à interoperabilidade entre dispositivos heterogêneos servindo diversos domínios de aplicação: adaptáveis, sensíveis ao contexto, autonômicas, com descoberta e gerenciamento de dispositivos, escaláveis e que possuam gerenciamento de grandes volumes de dados. Alguns esforços podem ser vistos em pesquisas em relação à utilização dos vocabulários semânticos em serviços da *Web*, demonstrando alguns problemas que persistem na detecção clara destes serviços. Os dispositivos embarcados interligados na *Internet* necessitam buscar conteúdos mais eficazes. Para realizar esta busca foi necessário analisar alguns trabalhos que utilizam a *Web* semântica a parti de um vocabulário capaz de interagir com os diversos dispositivos embarcados conectados na IoT. Sendo assim, este artigo apresenta um estudo da *Web* semântica em diferentes aplicações nos dispositivos embarcados que se conectam às aplicações ou coisas na *Internet*, assim como disseminar um vocabulário semântico para dispositivos embarcados baseados em arquitetura ARM e no arduino, e realizar uma simulação utilização a aplicação *Contiki*. Com isso, realizar um comparativo das principais características dos trabalhos relacionados com a *Web* semântica, vocabulário e dispositivos embarcados.

PALAVRAS-CHAVE: *web* semântica; *Internet*; embarcados; vocabulário.

STUDY OF *SEMANTIC WEB* IN EMBEDDED DEVICES FOR *INTERNET OF THINGS* (IOT)

ABSTRACT

The *Semantic Web* is a new way to assign a meaning to *web* content with a vocabulary. Currently there are several research into the construction of ontologies and vocabularies aimed at interoperability between heterogeneous devices serving different application areas: adaptive, context-sensitive, autonomic, with discovery and device management, scalable and that have managing large volumes of data . Some efforts can be seen in research on the use of *semantic* vocabularies in *Web* services, demonstrating some problems persist in the clear detection of these services. Embedded devices interconnected on the *Internet* need to search for more effective content. To perform this search was necessary to analyze some works that use the *Semantic Web* the left of a vocabulary able to interact with the various embedded devices connected to the IoT. Thus, this paper presents a *semantic web* study in different applications in embedded devices that connect to applications or things on the *Internet*, as well as disseminating a *semantic* vocabulary for embedded devices based on ARM architecture and Arduino, and perform a simulation using the *Contiki* application. With this, perform a comparison of the main features of the work related to the *Semantic Web*, vocabulary and embedded devices.

KEYWORDS: *semantic web*; *Internet*; embedded; vocabulary.

INTRODUÇÃO

A *Web* armazena uma enorme quantidade de dados e informações criadas por diferentes organizações, comunidades, usuários e outros, oferecendo assim uma variedade de serviços como comércio, turismo, notícias, ensino, movimentações bancárias e etc. Este rápido crescimento torna cada vez maior a quantidade de páginas indisponíveis, devido aos acessos simultâneos dificultando em alguns momentos a localização destas informações na rede.

Apesar de algumas deficiências, como a resposta de um conteúdo pesquisado com maior clareza, a *Web* possui um papel importante para a sociedade como um todo, e seu desenvolvimento contínuo constitui um desafio para a comunidade científica. De acordo com Lima (2004), o esforço na tentativa de integrar o entendimento semântico das informações ocorrerá o mais breve possível. Desta forma, várias iniciativas são buscadas por intermédio da criação de padrões, arquitetura de metadados, serviços de inferências e ontologias, sendo uma melhor forma de tornar as informações processáveis pelas máquinas.

Para entender melhor os conceitos abordados dentro da *Web* semântica, deve-se compreender que uma ontologia é um documento ou um arquivo onde estão definidas formalmente as relações entre os conceitos. Uma das definições mais conhecidas para ontologias é apresentada por Gruber (2009, p. 199) entende que:

Uma ontologia é uma especificação explícita de uma conceitualização. [...] Em tal ontologia, definições associam nomes de entidades no universo do discurso (por exemplo, classes, relações, funções, etc. com textos que descrevem o que os nomes significam e os axiomas formais que restringem a interpretação e o uso desses termos) [...]. Ou seja, ontologia é uma descrição (como uma especificação formal de um programa) dos conceitos e relacionamentos que podem existir para um agente ou comunidade de agentes.

Logo, é considerada como uma taxonomia formada de classes e subclasses de objetos relacionados entre si e com mais de um conjunto de regras de inferência que podem utilizar uma linguagem como a DAML (DARPA *Agente Mark-up Language*) desenvolvida como ontologia e inferência baseada em RDF (*Resource Description Framework*), conforme descrito em Lima (2004).

Esta arquitetura da *Web* semântica está incorporada no paradigma da IoT, visto que todas as camadas devem ser implementadas para que as tecnologias possam criar redes de "coisas inteligentes" que são encontradas no mundo físico, como tecnologias que tornam a IoT realizável, como a RFID (*Radio Frequency Identification*), WSN (*Wireless Sensor Network*) e WSAN (*Wireless Sensor and Actuator Network*).

Existe uma tendência em tratar a IoT em WoT (*Web* das Coisas), nos quais os padrões abertos da *Web* são empregados para prover o compartilhamento de informação e a interoperabilidade entre dispositivos, tendo alguns motivos que favorecem a WoT, como: a tecnologia integradora dos serviços *Web* que tem se mostrado indispensável na criação de aplicações distribuídas interoperáveis para *Internet*; coisas inteligentes (*smart things*), com servidores *Web* incorporados, que podem ser abstraídas como serviços *Web* e perfeitamente integradas em dispositivos embarcados.

O aumento da utilização de aplicações na *Internet* favorece para que diversas pesquisas sejam realizadas com aplicações integradas a sistemas embarcados. Assim, são necessários que sejam criados vocabulários semânticos que interajam com a *Web* e possam oferecer esta integração. Portanto, o objetivo deste artigo é apresentar um vocabulário semântico na utilização da *Web* semântica em dispositivos embarcados. Demonstrar que este vocabulário pode ser criado pela ferramenta Protegé e posteriormente testado numa simulação com o *Contiki* para sistemas embarcados e assim realizar a conexão destes com as aplicações das "coisas" através de API robustas. Além de processar o conteúdo semântico de forma rápida e também de efetuar buscas dos resultados com maior eficiência.

O restante do artigo está organizado em seis seções sendo: a Seção 2 apresenta uma fundamentação teórica dos conceitos necessários ao entendimento do trabalho, a Seção 3 apresenta os trabalhos correlatos, na Seção 4 é abordado a proposta de uma nova semântica. Já na Seção 5 os resultados analisados nos trabalhos correlatos. E por fim as conclusões com sugestões de trabalhos futuros são apresentadas na Seção 6.

FUNDAMENTAÇÃO TEÓRICA

Nesta seção são apresentados os principais conceitos empregados na utilização da *Web* semântica no funcionamento dos dispositivos embarcados para *Internet* das coisas. A presença generalizada de coisas e objetos que nos rodeiam e que são capazes de interagir refere-se a este conceito de *Internet* das coisas (IoT). Esta tecnologia ainda não alcançou a consciência das massas, mas, que tem vida longa.

O domínio tecnológico da IoT é construído por objetos inteligentes interligados, que pode direcionar mais o interesse para as tecnologias de comunicação e preservação ambiental. A IoT permite que "coisas" possam ser conectadas a qualquer hora e em qualquer lugar, conforme Atzori (2014).

Web Semântica

Com o crescente aumento da interação das pessoas com a *Internet* surge a necessidade de obtenção de informações, mas processáveis pelos computadores. Um simples computador consegue manter as informações em hierarquias, porém o raciocínio humano tem a habilidade

de integrar pequenas unidades de informações de forma aleatória. Com base nessa constatação, surgiu em 2001 a *Web* semântica publicada em um artigo na revista *Scientific American* denominado "*The Web Semantic: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities*", de acordo com Berners et al. (2014).

Para funcionar, a *Web* semântica deve ter o acesso às coleções de informações estruturadas e aos conjuntos das regras de inferências que poderão ser usadas para conduzir a um raciocínio automático. As linguagens utilizadas para as regras deverão ser tão expressivas quanto possíveis para permitir que se possa raciocinar sobre a *Web* tão amplamente quanto necessário, portanto, a *Web* semântica é vista como uma solução extraordinária para resolver problemas da *Web* atual.

Analisando a arquitetura da *Web* semântica, percebe-se que seus princípios são implementados em camadas de tecnologias e padrões *Web*, conforme se pode observar na Figura 1.

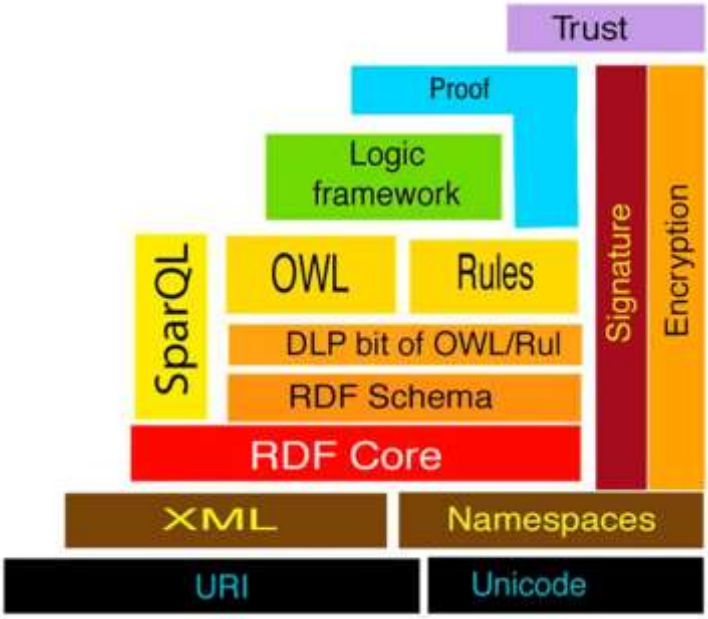

Figura 1. Estrutura de camadas da *Web* Semântica.

A seguir apresentam-se detalhes de cada componente da estrutura de camadas da *Web* semântica, apresentado na Figura 1.

Identificador – URI (*Uniform Resource Identifier*): define uma maneira padronizada de nomear recursos. Está ligado a algum tipo de arquivo publicado na *Web*, como: foto; página, mas seu uso na *Web* semântica é apenas identificar recursos;

Conjunto de caracteres – Unicode: padrão internacional para representação de caracteres, fornece um único número para cada caractere;

Sintaxe – XML (*Extensible Markup Language*): pode ser definida como uma linguagem de marcadores como a HTML (*HyperText Markup Language*) e foi desenvolvida para escrever dados;

Intercâmbio de Dados – RDF (*Resource Description Framework*): é uma linguagem que esta apta a prover um modelo de afirmações e citações em que se podem mapear os dados em qualquer novo formato;

Taxonomias – RDFS (*Resource Description Framework Schem*): é um framework que fornece meios para a criação de vocabulário RDF, onde é possível descrever novas classes e propriedades e relacionamentos entre elas;

Consulta – SPARQL (SPARQL *Protocoland RDF Query Language*): pode ser definido como o protocolo e linguagem para consultas a modelos RDF. Sua sintaxe é similar à linguagem SQL (*Structured Query Language*);

Ontologias – OWL (*Web Ontology Language*): é uma extensão do RDF, que inclui termos de vocabulário para descrever classes, fatos e relacionamentos sobre estas classes e características destas relações atrás de axiomas lógicos;

Regras – SWRL (*Semantic Web Rules Language*): são regras que visam novos conhecimentos em modelos RDF/OWL.

Dispositivos Embarcados

Um dispositivo pode ser classificado como embarcado quando interage continuamente com o ambiente a sua volta por meio de sensores e atuadores, segundo Ball (2005). Por exigir uma interação contínua com o ambiente, este tipo de dispositivo necessita que o projetista tenha um conhecimento em programação, sistemas digitais, noções de controle de processos, sistemas de tempo real, tecnologias de aquisição de dados atuadores e cuidados especiais na eficiência de estruturação do projeto e do código produzido.

Assim, a denominação "sistemas embarcados" (do inglês *Embedded Systems*) vem do fato de que estes sistemas são projetados geralmente para serem independentes de uma fonte de energia contínua, conforme Ball (2005). As principais características de classificação desses sistemas são: capacidade computacional e independência de operação. Outros aspectos relevantes dependem dos tipos de sistemas, modos de funcionamento e dos itens desejados nas aplicações embarcadas.

A arquitetura ARM (*Advanced Risc Machine*) tem sido amplamente utilizada nos últimos anos, devido à simplicidade e a eficiência das instruções de seus microcontroladores. Assim, está arquitetura são amplamente utilizados em aparelhos eletrônicos modernos, como *smartphone*, *tablets* e etc, e os sistemas embarcados, cada vez mais, têm se tornado atraente para o desenvolvimento de novas tecnologias, pois é dedicado a tarefas específicas, podendo-se um projeto ser aperfeiçoado reduzindo tamanho, recursos computacionais e custo do produto.

Um exemplo de dispositivo que utiliza a tecnologia embarcada é o arduino que está conectado a uma rede Ethernet por *wireless* para acessar a *web* e assim utilizar uma API. Um exemplo, de aplicação embarcada usando arduino conectado a uma rede *wireless* é mencionando por Franz et al. (2011), no qual o projeto se focaliza no monitoramento de agentes poluentes em cinco locais diferentes no noroeste de Indiana, Estados Unidos.

TRABALHOS CORRELATOS

Nesta seção são elucidados alguns dispositivos embarcados que utilizam a *web* semântica para a IoT e suas principais características. Estas estão relacionadas com as exigências que foram vistas anteriormente e com alguns trabalhos que pesquisados e estudados correspondem ao assunto tratado.

Cosm

É uma ferramenta desenvolvida para gerenciar diversos dispositivos através de recursos REST *ful* (*Representational State Transfer*) em uma plataforma como serviço para a IoT. De acordo com Franz et al. (2014), no processo de enriquecimento do conteúdo semântico para dispositivos embarcados conectados a IoT, este sistema integra um dispositivo como entrada e gera uma representação XML do dispositivo com uma descrição de texto em

linguagem natural. O XML *mark-up* é utilizado como uma representação semântica do dispositivo embarcado usado como entrada para vários sistemas, como por exemplo, para tarefas de engenharia e sistemas automáticos de geração 3D.

Como um projeto prático pode-se citar um arduino com sensores que recolhem dados de agentes poluentes em cinco diferentes localidades, conectados a uma rede *wireless* transmitindo informações para um servidor *web* que realizará a correlação dos dados com o servidor Cosm *semantic*amente.

Comunicação Móvel 3G

A agregação dos dados entre diferentes tipos de Redes de Sensores Sem Fio (RSSF) e transmissão de longa distância através dessas redes de dados móveis favorece o envio das informações adquiridas por qualquer nó sensor na RSSF a qualquer lugar do mundo.

O projeto de um *gateway* 3G embutido com base em na arquitetura ARM, conforme descrito por Zhang et al. (2014), realiza a agregação de dados heterogêneos com a estreita combinação de RSSF e rede 3G. Possui como vantagens: a elevada confiabilidade; forte capacidade de processamento de dados; taxa de transferência de dados rápida; e capacidade de resposta em tempo real. Segundo o autor no futuro próximo esta fusão melhorará a qualidade de aplicações IoT e proporcionará um futuro com dispositivos mais inteligentes para a vida das pessoas.

Gateway IoT

Como surgimento da IoT, um novo tipo de equipamento de rede foi inventado, chamado de *gateway* IoT, cujo objetivo é resolver a heterogeneidade entre as diversas redes de sensores sem fio e de comunicação ou *Internet* móvel, conforme Zhu et al. (2014). Portanto, as questões-chave da execução de um sistema IoT *Gateway* é a resposta de diferentes redes e diversidade de protocolos no sensor WSN e redes de telecomunicações tradicionais. Além disso, é necessária a construção de um conjunto de instruções e normas identificados para todos os *gateways* IoT, a fim de perceber a funcionalidade de gerenciamento e controle IoT.

Este projeto de um *gateway* IoT é baseado em alguns requisitos como: transmissão de uma rede por 2G/3G; comunicação por redes DSL; conversão de protocolo IEEE 802.15.4/ZigBee; e controle e gerenciamento de nós dos sensores de redes.

Arquitetura de Protocolos

Os dispositivos são esperados para “superarem” os humanos e se tornarem os maiores geradores e receptores de tráfego. Muitos ligados à *Internet*, como *smartphones*, laptops e suas várias combinações, possuem a tecnologia RFID que tem sido apontada como uma das tecnologias que permitem a conexão com *Internet* das coisas, assim como a robótica e a nanotecnologia.

A prevalência da *Internet* como meio de comunicação e a disponibilidade de custo de ferramentas eficazes, favoreceu o desenvolvimento de uma grande variedade de aplicações e protocolos de rede. De acordo com Silverajan (2009), tais desenvolvedores implementaram o *software* de rede e aplicações dos protocolos de comunicação. Com isso, os protocolos tendem a impactar no tamanho, desempenho e robustez das aplicações resultantes. Por isso, é importante entender como exatamente estes protocolos de comunicação são desenvolvidos.

O desenvolvimento de um *framework* que entenda o conteúdo semântico na *web* e esteja interligado com cada protocolo de aplicação é fundamental para que se obtenha um desempenho significativo nas aplicações de IoT, de acordo com Zhexuan et al. (2010).

Emulator QEMU

Uma arquitetura de sistema usando vários tipos de virtualização embarcada foi proposta em Kovacshazy et al. (2013). Mantendo questões de construtibilidade e operabilidade, além de avaliar a arquitetura do sistema proposto utilizando um protótipo construído de uma única placa de computador, sensores e atuadores. Além disso, utiliza *software* de código aberto, como Linux e QEMU.

Para a virtualização do processador é utilizado o emulador QEMU que possui como principal vantagem da emulação em modo usuário, permitindo que execute o programa com maior velocidade de processamento, passando a responsabilidade de entrada e saída para o *Kernel* do Linux.

Steer

Um sistema computacional implementado para dar suporte a mecanismos de descobertas de serviços, além de ferramentas e serviços da *Web* semântica, foi criado composto do STEER (*Task Semantic Editor execution)* e PIPE (*Pervasive Instance Provision Environment*). Um cliente *Web* com uma interface em *Java* classifica os serviços com base nos seus dados de entrada e nas saídas semânticas. Através de um gerenciador GUI os serviços do PIPE são facilitados para os usuários para que desta forma possam executar serviços *Web* para tarefas comuns.

A OWL é a linguagem ontológica utilizada pela semântica desta aplicação com modelagem poderosa é suficiente para descrever naturalmente muitos domínios. Assim, facilita os metadados da RDF (Resource Description Framework) e OWL descreverá de forma eficaz todos os tipos de recursos da *Web* para ambos os seres humanos e programas, proposto em Masuoka (2003).

Middleware IoT

Adicionando a tecnologia RFID em objetos foi dito que teríamos a IoT (*Internet* das Coisas), embora tornou-se uma realidade abrangendo em outros dispositivos como sensores, dispositivos móveis, entre outros. Considerando que existe uma série de desafios como: a escala de análise de milhares de dispositivos; a heterogeneidade entre os dispositivos integrando muitos componentes; a topologia desconhecida, com várias características e serviços diferentes; os metadados incompletos desafiando o uso da semântica; e a resolução de conflitos com as mudanças incompatíveis com os ambientes de acordo com Zhexuan et al. (2010).

A solução seria talvez uma arquitetura de middleware para IoT que consiste em três módulos conforme a Figura 2.

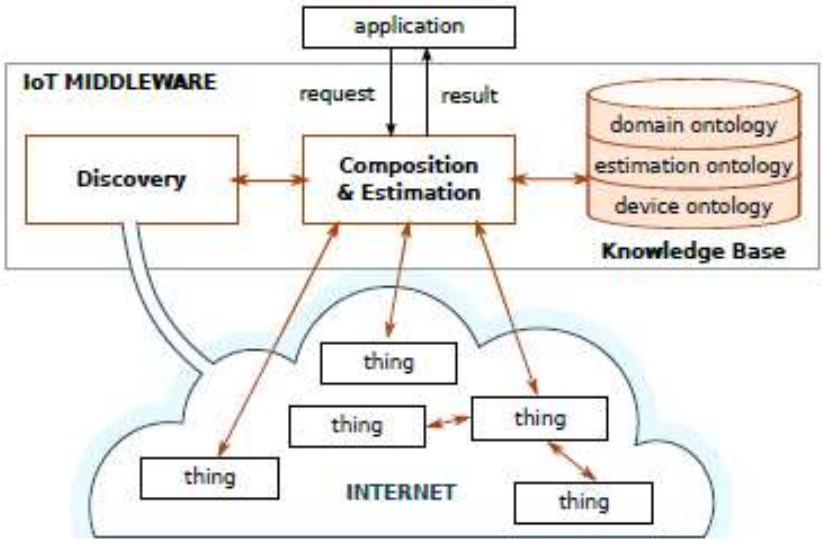

Figura 2. Arquitetura de *middleware* para IoT.

Neste *middleware* o pedido ou um serviço faz uma detecção, que obtém uma composição ou estimação do objeto manipulado. Para resolver a solicitação, este módulo acessa o módulo de descoberta (*Discovery*) e faz o conhecimento da base, realizando a criação de uma composição de serviços de acordo com Hachem (2011).

Ontonym

Os sistemas invasivos possuem a necessidade de interpretar grandes quantidades de dados a partir de muitas fontes. Um modelo de contexto foi criado para apoiar desenvolvedores que trabalham com esses dados, fornecendo dessa maneira um compartilhamento com o ambiente em que se baseiam essa interpretação. O *Ontonym* foi desenvolvido para abordar as propriedades temporais de contexto desses dados como requisito. Então, o critério para que funcione é utilizar um conjunto de ontologias para representar os conceitos fundamentais em computação pervasiva. Avaliando as ontologias no domínio da computação pervasiva através da combinação de técnicas reconhecidas a partir da literatura de acordo com Stevenson (2009).

As ontologias são descritas no *Otonym* para fornecer a interoperabilidade semântica entre os diferentes sistemas descrevendo a computação pervasiva no núcleo da sua literatura. Os conceitos que o envolvem para realizar uma comparação com as ontologias da computação pervasiva são: o tempo, usando a linguagem ontológica OWL-Time para representação temporal; a localização, representando a posição física gerada por um GPS; as pessoas, onde são coletadas as informações pessoais e das redes sociais; os sensores ontológicos gerando os dados.

PROPOSTA DA SEMÂNTICA

Neste artigo propomos desenvolver um conteúdo vocabulário semântico que obtenha um melhor resultado para os dispositivos embarcados que utilizam conexão sem fio para IoT.

O teste desta conexão sem fio com a IoT será realizado através de um simulador chamado *Contiki* encontrado no site www.*Contiki*-os.org, sistema operacional de código aberto, desenvolvido para IoT pelo Instituto de Computação da Suécia de acordo com Barros

(2014). As principais características do *Contiki* são o carregamento e descarregamento dinâmico de código em tempo de execução e programação concorrente no *Kernel*.

Com este simulador são utilizadas bibliotecas para criar motes que simulam a baixa capacidade de processamento e a economia de energia dos sistemas embarcados como pode ser observado na Figura 3. Em seguida, a ideia é utilizar uma ferramenta, chamada de *Protegé*, para desenvolver o novo vocabulário semântico onde os diversos dispositivos embarcados que se comunicam com aplicações da IoT e possam trazer a correlação dos conteúdos da *web* de maneira mais robusta e eficaz, facilitando assim a busca dos usuários na utilização das informações nos aplicativos com a IoT.

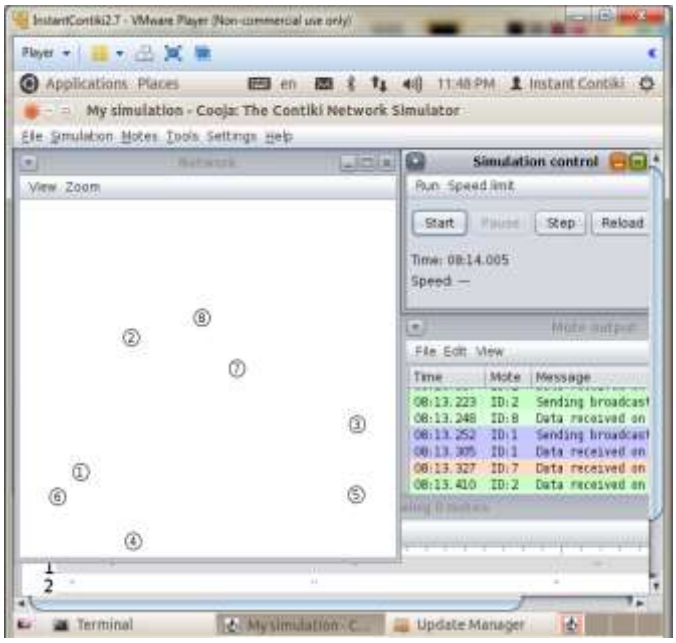

Figura 3. Interface do simulador *Contiki*.

Esta plataforma foi desenvolvida pelo grupo de pesquisa *Stanford Medicina Informatics* da escola de medicina da Universidade de *Stanford* e permite construir ontologias, que são os vocabulários utilizados na *web* semântica.

Baseada na linguagem Java, é extensível e conta com uma forte comunidade de desenvolvedores, acadêmicos e usuários, os quais se utilizam do *Protegé* para soluções de conhecimento de áreas diversas como a biomedicina. Neste artigo, ele foi escolhido como ferramenta de criação da semântica devido a vários fatores, sendo: um editor de ontologias de código aberto, onde suporta a edição via *Web Client* ou via *Desktop Client*, ou seja, o acesso pode ser feito via *web* ou aplicativo instalado na máquina, permitindo o desenvolvimento de ontologias em vários formatos, como: OWL, RDF e XML *Schema*. Permite também a utilização de vocabulários já existentes durante a criação da própria ontologia e a exportação da ontologia para o formato desejado, além de possuir uma interface gráfica de fácil manipulação.

A interface gráfica provê acesso à barra de menus e à barra de ferramentas, além de apresentar cinco áreas de visualização (*views*) que funcionam como módulos de navegação e edição de classes, atributos, formulários, instâncias e pesquisas na base de conhecimento, propiciando a entrada de dados e a recuperação das informações. A interface da plataforma *Protegé* pode ser visualizada na Figura 4.

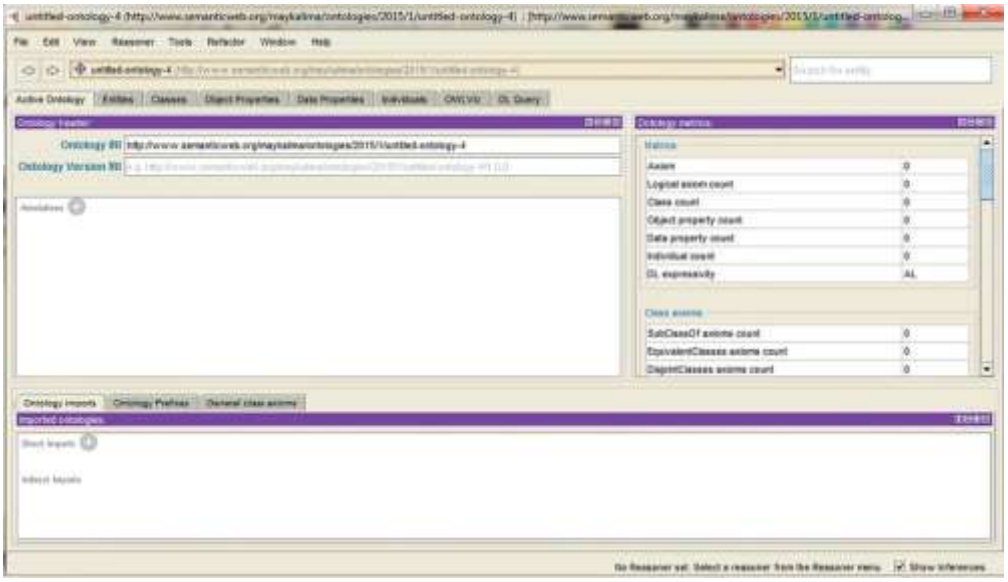

Figura 4. Interface da plataforma Protegé.

A semântica a ser criada deverá seguir os padrões definidos anteriormente para que possam ter maior interação com aplicações IoT e com o dispositivo embarcado. Toda a estrutura a ser montada deverá possuir sistemas abertos, como o Linux, na criação de um servidor *web* para a aplicação de teste.

A arquitetura embarcada a ser utilizada pode ser um processador ARM para os testes com o vocabulário semântico, integrado a uma aplicação *web* como o *Cosm*, para interagir com o usuário na comunicação e pesquisar o conteúdo semântico desejado. Diminuir a quantidade da intervenção humana na busca das informações, agilizar a análise semântica e executar os comandos solicitados pelos usuários com melhor compreensão são as métricas necessárias para obtiver um resultado eficaz nas aplicações de IoT nos sistemas embarcados.

Um exemplo de utilização de vocabulário padrão, seria a utilização de alguns objetos do lado do cliente com um serviço que fazem por meio de uma classe *proxy* que "imita" as chamadas de métodos do serviço *web*. Trabalhando com o *proxy* em vez de escrever mensagens SOAP (*Simple Object Access Protocol*) diretamente. A classe *proxy* gerencia a construção, envio e recebimento de mensagens SOAP e a linguagem OWL-S tem o propósito de possibilitar a automação da descoberta, invocação e composição de serviços *web* através de descrições semânticas. Os axiomas de OWL-S são definidos em OWL e o OWL-S possui descrições de classes, hierarquia de sub-classes e definições de tipos e relações entre as classes. A descrição das funcionalidades de um serviço é referenciada por uma ontologia de perfil da OWL-S externamente com os conceitos definidos em OWL na ontologia de domínio ao qual pertence o serviço *web* em questão.

Analisando os trabalhos existentes nesta área podemos constatar os problemas das aplicações que envolvem IoT com os dispositivos embarcados e assim chegar a uma necessidade de melhorar a *web* semântica para comunicação entre os mesmos criando novos vocabulários que possam interagir com diferentes dispositivos conectados às coisas. O simulador *Contiki* gerará alguns testes para avaliar o processamento e consumo de energia das aplicações utilizadas por estes dispositivos.

RESULTADOS E ANÁLISES

A utilização de ferramentas como o *Cosm* e o *Contiki*, é de suma importância para testar aplicações que envolvem semântica em dispositivos embarcados, por serem de código

aberto, e do Protegé para criação de vocabulários que possam se comunicar com estes dispositivos.

De acordo com a análise obtida no trabalho de Franz et al. (2014), na utilização dos sensores MQ131 e MQ7 , os dispositivos embarcados conectados à IoT podem ser enriquecidos com a semântica construída, tendo os resultados da tarefa de classificação (poluído vs. não-contaminada), e extraída as *meta-tags* do servidor da aplicação Cosm. Desta maneira, produz uma representação semântica, e uma descrição do dispositivo embutido na linguagem natural.

Estas características que podem ser utilizadas para prever se a área onde o dispositivo está localizado é poluído ou não contaminado, conforme pode se observar no código da semântica (ver Figura 5) construída incorporada no dispositivo embarcado.

Observando a Figura 5, destaca-se que as classes resultantes por dispositivo são combinados com a *tag* extraída do arquivo baixado do servidor *Cosm* html para criar uma lista de informação associados ao dispositivo, gerando uma representação em XML da iteração do sistema, dos sensores, das inferências semânticas e do dispositivo microcontrolador.

Alguns dos trabalhos necessitam de maiores testes de comunicação. Como por exemplo, pode-se citar a estrutura de comunicação 3G, onde necessita que as RSSFs estejam corretamente configuradas para não afetar o desempenho na transmissão dos conteúdos para as aplicações na *Internet*.

```
<?xml version="1.0" encoding="ANSI_X3.4-1968"?>
<embedded_device>
<semantic>
   <inference>
   <location at_location="university"/>
   <location at_location="factory"/>
  ...
</inference>
<sensor orderid="3" MarkableID="D001">
            <class is_fixed="1" polluted_site="1" >
<type f="MQ7"/>
</class>
</sensor>
...
<raw_features>
      <feature f=pressure value="30">
      <feature f=dew_point value="28">
      <feature f=relative_humidity value="0.45">
      <feature f=time value="12">
</raw_features>
...
</semantic>
</embedded_device>
```

Figura 5. Representação semântica incorporada

No trabalho pesquisado por Silverajan et al. (2009), envolve os protocolos de comunicação utilizados nas tecnologias RFID. No entanto, possui uma arquitetura de *framework* chamada de DOORS que implementa um subsistema CORBA, alguns protocolos de transporte como RTP (*Real-Time Transport Protocol*) e RTCP (*Real-Time Transport Control Protocol*), bem como o UDP (*User Datagram Protocol*) e TCP (*Transmission Control Protocol*) nas classes auxiliares para IPv4 e IPv6. Esta integração com o *frameware*, *middleware* de código aberto e tecnologias RFID integraram com a IoT obtendo maior produtividade e eficiência no resultado da capacidade de interação com as “coisas”. Este por sua vez, oferece alguns problemas na utilização de alguns protocolos de comunicação, pois dependerá se o mesmo suportará um vocabulário criado para a *web* semântica mais interativa.

Ainda tratando-se dos trabalhos aqui analisados, primeiramente, nenhum deles incorpora mecanismos de que garantam que os vocabulários semânticos utilizados nas

descobertas dos serviços supram os termos suficientes a realizar uma comparação livre de falhas. Caso o sistema, que utiliza o vocabulário para a descoberta de um serviço, que estiver sendo testado, dê um resultado não coerente, ocasionará em um retorno ao usuário inadequado a solicitação realizada.

Existem algumas diferenças evidentes quando comparamos estes trabalhos. Na Tabela 1 é possível visualizar a comparação entre características analisadas nos trabalhos relacionados neste artigo.

Tabela 1. Comparação entre arquiteturas para vocabulário semântico na IoT.

Caracaterística	STEER	Middleware IoT	Ontonym
Análise de similaridade semântica	X	X	X
Classificação de serviços	X	-	-
Simulação do vocabulário	X	-	-
Linguagem Ontológica	X	X	X
Testes com dispositivos da IoT	-	-	-

Fonte: Autoria própria

CONCLUSÕES

Este trabalho realizou o estudo da *web* semântica em dispositivos embarcados utilizados na IoT. A partir desse estudo pode-se perceber que a *web* semântica e sua aplicação em sistemas embarcados ainda são pouco difundidas. Atualmente carece de usuários, experiências e informações sobre o mesmo. Por outro lado, proporcionou um conhecimento e aprofundamento sobre o assunto. Dessa forma, grande parte do tempo dedicado a este estudo foi voltado para a análise das técnicas, ferramentas e arquiteturas criadas que são utilizadas com aplicações IoT para oferecer melhor desempenho na obtenção dos conteúdos buscados pelos usuários nas aplicações das coisas.

Novos cenários para integração de dados tornam-se cada vez mais inter-relacionados, mas em contrapartida, traz novos desafios, como por exemplo, a aplicação das linguagens próprias, RDF e SPARQL para o padrão da semântica. Trabalhos futuros sugeridos são aplicados a novos vocabulários semânticos e metodologias integradas às redes sem fios e aos dispositivos embarcados gerando conteúdos mais claros nas aplicações da IoT.

Atualmente as arquiteturas como: ARM e a plataforma Arduino fornecem a capacidade de monitoramento do ambiente no mercado interligados a IoT. É importante ressaltar que o Arduino, por exemplo, é uma plataforma de prototipagem eletrônica *open-source* baseada em *hardware* e *software* flexível e de fácil utilização. Também possui um microcontrolador que consome pouca energia e assim fornece ao usuário o controle completo do seu *hardware*. Outros microcontroladores possibilitam uma forma rápida de desenvolvimento de protótipos como as arquiteturas ARM utilizadas na maioria dos sistemas embarcados. Portanto, desta forma observa-se que a *web* semântica pode ser utilizada em qualquer dispositivo embarcado, permitindo assim sua expansão na interligação com as coisas inteligentes.

REFERÊNCIAS

ATZORI, L.; IERA, A.; MORABITO, G. The *Internet of Things*: A survey. Itália, 2010. Disponível em:

http://www.science.smith.edu/~jcardell/Courses/EGR328/Readings/IoT%20Survey.pdf. Acessado em 15 de setembro de 2014.

BALL, S. Embedded Microprocessor Systems: Real World Design, 3rd edition: MCPros, EUA, 2005.

BARROS, E. B. C.; RIBEIRO, A. de R. Jemadarius: Uma *Web*-API Autoconfigurável para *Internet* das Coisas. Universidade Federal de Sergipe, São Cristóvão, 2014.

BERNERS, T. L.; HENDLER, J.; LASSILA, O. The *Semantic Web*. Magazine Scientific American. 2001.

FRANZ, D. R.; CALLIX, R. A. *Semantic* Content Enrichment of Sensor Network Data for Environmental Monitoring. *In Proceedings of the Twenty-Seventh International Florida Artificial Intelligence Research Society Conference*. Purdue University Calumet. 2014.

GRUBER, T. R. What is an ontology? Springer-Verlag, 2009. *Knowledge Acquisition*, 5(2):199-220.

HACHEM, Sara; TEIXEIRA, Thiago; ISSARNY, Valeria. Ontologies for the *Internet of Things*. INRIA Paris-Rocquencourt. HAL Id: hal-00642193. 2011

KOVÁCSHÁZY, T.; WACHA, G. System Architecture for *Internet of Things* with the Extensive Use of Embedded Virtualization. Inter-University Centre for Telecommunications and Informatics Debrecen, Hungary. Cognitive Infocommunications (CogInfoCom), 2013 IEEE 4th International Conference.

LIMA, J. C. de; CARVALHO, C. L. de. Uma Visão da *Web* Semântica. Instituto de Informática da Universidade Federal de Goiás, Goiás, 2004.

MASUOKA, Ryusuka; et al. Ontology – Enabled Pervasive Computing Applications. University of Maryland. 2003 IEEE.

SILVERAJAN, B.; HARJU, J. Developing Network *Software* and Communications Protocols Towards the *Internet of Things*. Department of Communications Engineering Tampere University of Technology. Comsware 2009.

STEVENSON, Graeme; et al. Ontonym: A Collection of Upper Ontologies dor Developing Pervasive Systems. 2009.

ZHEXUAN, S.; CARDENAS, A. A.; MASUOKA, R. *Semantic* Middleware for the *Internet of Things*. Fujitsu Laboratories of America, Inc. IEEE. 2010.

ZHANG, Y. Y. The Design of Embedded 3G Gateway for *Internet of Things*. School of Communication and Information Engineering Chongqing University of Posts and Telecommunications Chongqing, China. 5th International Conference on BioMedical Engineering and Informatics (BMEI 2012).

ZHU, Q.; WANG, R. IOT Gateway: BridgingWireless Sensor Networks into *Internet of Things*. School of *Software* and Microelectronics, Peking University, Beijing, China. IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (IFIP 8th).